マニュアルバイブル

使いやすい
ユーザー・マニュアルの書き方

エドモンド.H.ワイス・著　小林敦・訳

USABLE USER MANUAL

Edmond H. Weiss
Atsushi Kobayashi

啓学出版

マニュアル・バイブル

――使いやすい ユーザー・マニュアルの書き方――

エドモンド.H.ワイス・著　小林敦・訳

啓学出版

for Beverly

ワードスター®は、マイクロプロ社の登録商標です。

HOW TO WRITE A USABLE USER MANUAL

Edmond H. Weiss

日本語版への序

コンピュータやハイテク製品の需要が高く、供給がそれに追い付かない場合、生産者はマニュアルの品質にはほとんど気を遣わないのが通例である。しかし競争が激化し、ハードウェアやソフトウェアがだぶついてくると、生産者や販売者は、客の目をひき、気に入られる方法はないかと躍起になる。コンピュータや他のハイテク製品の販売商戦が激しくなればなるほど、マニュアルの重要性は高くなる。

本書がアメリカで出版されてから2年の間に、マニュアル作成の問題に対する意識は、劇的に向上してきた。マニュアルを“飾りもの”ぐらいに考えていた企業も、今では製品にとって必要不可欠な拡張部分、つまり周辺機器の1つといってもいいものとして、マニュアルを認識している。

今日、大手ハードウェアメーカーの多くは、“有用性テストのための研究室”を設置し、製品のみならず、マニュアルや他の印刷物の使いやすさに関してもテストを行なっている。

しかしながら、マニュアル作成にどんなに関心を示し、情熱を傾けても、事態はいっこうに改善されていない。ほとんどのマニュアルは、相変わらず使いにくく、理解に苦しむものである。いまだにマニュアルおよびコンピュータ関連の出版物の大半が、必要な訓練をほとんど、あるいはまったく受けず、資質も充分でないような者によって書かれている。そして相変わらず、信じがたい数のマニュアルが、自分の仕事が嫌いで嫌いで仕方がないという雇われ人によって書かれている状態である。

その上、有用性をテストすることが、いつも有効な解決策であるとは限らない。本書が示す通り、マニュアルの第一稿が完成してしまってからでは、もっとも深刻な欠陥に対処するには遅すぎるのである。

したがって、日本の読者への私のメッセージは単純である。よいマニュアルを書くことができるのは、気を配るライターのみであるということだ。こうしたライターは、言葉に気を配るに違いない。図表や印刷に気を配るに違いない。そして、これがもっとも重要だが、彼らはマニュアルの読者の利益に気を配るに違いない。

Edmond H. Weiss, Ph. D.
April 1987

目　次

付録

序：誰のために書かれたか

本書は、マニュアル作成に何らかの形で関与するすべての人、すなわち企画、執筆、編集、出版などに携わる人はもとより、それらを管理する人をも含めたあらゆる人々に捧げるために書かれたものである。しかし、本書を一番必要としているのは、既存のマニュアルのほとんどに失望している人々(書いた本人も含む)であろう。本書は、そのような読み手にとってマニュアルがより適切で、なじみやすく、しかも読みやすいものになるように、一連の発想、原則、手法などを提供するものである。

●マニュアルを考えるときがきた！

マニュアル作成について真剣に考える時期がきているようだ。特に、ユーザーやオペレーターを対象としたソフトウェアのマニュアル作成に関しては、それをテーマにした本が何冊か新しく出版されたり、大学院の取得教科になっていたりする。マニュアル作成とは、コンピュータ科学はもちろん、技術訓練指導、グラフィック・アート、レトリックや作文法の伝統的な手法などさまざまな分野の技法・技術を結集したものという意味から、明らかに1つの新しい専門分野であるといえるだろう。

こうした機運が高まってきた原因は明らかだ。ここ10年ほどの間に、コンピュータ、ソフトウェア製品、およびそれらの周辺機器の数は爆発的な増大を見せた。そしてほとんど例外なく、これらのシステム、機器、付属品などは、そのユーザーやオーナーのために、何らかの文献を必要としてきた。

このようなマニュアルの需要が量的にどれだけ増大したかはともかく、こうした状況が、マニュアルの対象となる読者層に大きな変化をもたらしたことは見逃せない事実である。かつて(ほんの10年ほど前だが)、オペレーション・マニュアルの読者は、科学者、技術者、あるいは熟練者が主だった。しかしながら今日、コンピュータ・テクノロジーに関する訓練をほとんど、あるいはまったく受けていないビジネスマン、難解で込み入ったマニュアル（’70年代の半ばには、これが常識だった！）を使いこなせる見込みのない事務員や作業員などが読者の主流なのである。

いつの時代にも、よくできた、ほとんど完璧ともいえるマニュアルはあるにはあったが、それらは極めてまれで例外的な存在だった。かつて、技術者や高度な訓練を受けた専門要員が主にコンピュータを操作していた時代には、使いやすく読みやすいマニュアルの必要性はあまり

なかった。技術職に就いている人々は、マニュアルに難解さを求め、もつれた糸を解きほぐすようにして、自分の方法を学び取っていくのが常だった。しかし今日では、不親切なマニュアルと格闘しようというユーザーはまれである。実際、それをやろうとしてもできないのが現実なのである。

●マニュアルの基本条件とは

興味深いことに、よいマニュアルを書くための基本的な条件は、昔から変わっていない。マニュアルは、まず何よりも**見たいときにいつでも利用できなければならない**。また見直しやテストを重ねた結果として、**正確かつ完璧**な内容をもつようにならなければならない。そして、**正しい言葉**で書かれ、**明白で読みやすい**形に編集されなければならない。

これらの大原則は普遍的なものだが、変化したものもある。それは、読む側の許容度と評価基準である。今日のユーザーは、たとえ専門家や技術者であれ、読みにくく分かりにくいマニュアルに、あまり寛容ではなくなっている。昔のジョークにこういうのがある。「他になすべき手立てがなくなったら、マニュアルを読め。」もはや笑い話ではすまない。システムやソフトウェアの製作者は、効果的なマニュアルを作ることが、市場競争に勝つためのキーポイントになる場合が多いということを、ようやく気付いたようである。コンピュータ業務の統轄者たちも、正確なマニュアルとガイドブックを備えることが、データ処理部門の生産性の飛躍的向上につながるということを、ようやく理解し初めたようである。また、アプリケーション開発の初期の段階でユーザー用の文書を作成することが、その製品自体の質的向上に役立つということを、体験的に学び取っているシステム・アナリストもいるようである。

しかし、マニュアルの質を改善したいと思う気持ちだけでは充分ではない。今日でさえ、ほとんどのマニュアルが、何ら特別な訓練を受けていない人々の手によって書かれている。私の察するところ、代表的なユーザー・マニュアルは今でも、選ばれたので仕方なく、というシステム・アナリストやプログラマーの手になるもののようである。だが、それ以外のマニュアルは、教育学や人文科学の分野から最近採用された才能溢れる人々の手によって書かれることが多いようだ。彼らの参加はコンピュータ業界にとって極めて有益ではあるが、ユーザー用文書の作成技法に関する彼らの準備・研究はまだはなはだ不十分である。実際のところ、マニュアルの製作・出版に長年かかわってきたエキスパートでさえ、'60年代以来の、一つの枠にすべてをあてはめようとする古い文書作成基準はもはや通用しなくなった、と嘆いているのである。

●本書の目指すもの

したがって、この本の目指すところは、上述のすべてのグループの人々、そしてまたマニュアルの企画、設計、作成に関する普遍的な原理を得たいと願っている人々の助けとなることである。本書には、この分野への新入生がやっかいな仕事にもひるまず立ち向かうことができるような、さまざまなアイデアが盛り込まれているはずである。(実際彼らは、込み入った製品やシステムを、経験のない、ときにはまったく専門外の読者に、うまく説明しなくてはならないのである。)また、経験豊かなテクニカルライターやエディター、文書作成の専門家にとっては、マニュアルをどのように改善したらよいかを分析し、またその改善に費用がいくらかかるかを頭の硬い関係者や上司に納得させるための１つのツール（道具）となるであろう。本書において、何よりも興味深くしかも実用的であろうと思われること、またそうあらんことを望むことは－これが本書の中心テーマでもあるのだが－次のようなことである。**マニュアルを作成していく過程は、コンピュータの複雑なプログラムを開発していく過程と非常によく似ているのである。**

注：本書は、他と比較していえば一般的ではない書式と形式で書かれている。もっと明確にいうならば、多くのマニュアルがこうであって欲しいと思う書式と形式で書かれている。もちろん本書は、マニュアルでは「ない」。したがって、このスタイルは風変わりにも不適切にも見えるかもしれない。出版社と私は、そうしたリスクをあえて負うことにした。他の多くの場合と同様、マニュアル作成においても、実例こそが最良の教師である、と信じているからである。

謝　辞

測り知れない数の人々が、本書の概念形成に手を貸してくれた。ソフトウェアやマニュアルの多くのスペシャリストの作成した文書から得るところが大だった。そしてまた、私のクライアントたちからも多くを学んだ。彼らは、何が問題なのかを私に教えてくれた。特に NCR 社の Stephen Bean、Kenneth Helms、Madeline Flynn、Michael Bartlett の各氏に深く感謝しなければならない。彼らは問いを投げかけてくれたのみならず、少なからぬ答えをも提示してくれた。

本書の一部は、そのままの形ではないが、*Data Training* 紙に一度掲載されたものである。第二章の一部分が1984年２月号に掲載された。これらは DATA TRAINING, Warren/Weingarten 社の許可を得て使用している。

DATA TRAINING, Warren/Weingarten, Inc., 38 Chauncy St, Boston MA 02111, (617) 542-0146.

その他、企業のマニュアルからの抜粋は下記の人々に著作権使用の許可を得たので、謝意を表しておきたい。

Jennie Balcom, Pioneer Data Systems, *VAX MAIL User's Guide*
McDonnell Douglas Computer Systems Company, *Introduction to REALITY*
Wharton Econometric Forecasting Associates, Inc., Philadelphia, PA, *Tables Documentation : A Teaching Introduction*

最後に、私が ISI Press 社の Maryanne Soper 氏に道をつけていただいた幸運な著者の１人であることを表明したい。

凡　例

・訳文中でのゴシック文字は、原文中でのゴシック文字に対応している。
・訳文中で「　」囲みになっている文字は、原則として原文中のイタリック文字に対応しているが、強調のために訳者が任意に追加したものもある。
・訳文中で“　”囲みになっている文字は、原則として原文中でも“　”囲みになっている文字に対応しているが、強調のために訳者が任意に追加したものもある。
・訳文中で(　)囲みになっている文字は、原則として原文中でも(　)囲みになっている文字に対応しているが、補足のために訳者が任意に追加したものもある。
・訳文中での一は、原則として原文中の一に対応しているが、日本語の性質上、訳者が任意に省略、あるいは(　)による置換えを行なっているものもある。
・訳文中で●で始まっている小見出しは、原文にはなく、読みやすさを考慮して訳者が任意に追加したものである。
・訳文中の下線は、原文にはなく、重要と思われる部分に訳者が任意にひいたものである。
・原注は、原文にはなく、訳者からの著者への問合わせに対し、著者から送られてきた解答を訳出したものである。
・訳注は、読者の便宜を考え、訳者が任意に付けたものである。

第 I 部

マニュアルを科学する

第1章

役割：マニュアルは何をするか

1.1　ユーザーとは何か？　マニュアルとは何か？

ユーザーとは、満足させせなくてはならない存在である。さまざまな団体や個人が、独自の利益を追求するためにいくつかの目標を設定して、コンピュータ・テクノロジーを購入し、またそれを開発している。たいていの場合、ユーザーは、設定した目標が必ず達成され、利益が得られるものと確信している。マニュアルは、こうしたユーザーが、十二分な有用性と、思い通りの利益を獲得できるよう手助けする重要な情報ツールである。

●ユーザーを満足させよ

「ユーザー」とは何かを定義することは、冒険的な試みである。慎重な言い方をするなら、「ユーザー」とは、**データ処理の単なる結果ではなく、それによってどのような成果が期待できるのかということの方に、より多くの関心を示す存在**なのだと定義することができるだろう。

ユーザーとは「満足」させなければならない存在だという場合、それは妥当な労力と費用をかければ、目的は必ず達成されると信じているのがユーザーだという意味である。途中経過よりも最終結果、単なる結果よりも成果、というわけである。つまり、ユーザーにとっては、システムがどのような「働き」をしているのかよりも、自分の個人的な利益に対して、システムがどのような「働き方」をしてくれるのかの方が重要なのである。

確かにユーザーは、コンピュータの内部の働きにも興味を示すようになるだろう。しかし、それは次のような基本的な概念をくつがえすものではない。すなわちユーザーとは、自分の使用している特定のデバイスより大きな何ものかを、その内側ではなく、外側に望むものであるということだ。コンピュータの世話にならずに、欲しいものを手に入れる経済的な方法があるとしたら、彼らは迷わずそちらを選ぶだろう。

なぜこの点を強調する必要があるかといえば、コンピュータの技術開発やそれに付随する文書作成に携わる人々の多くに、テクノロジーの開発自体を最終目標とみる傾向があるからである。また、そうした傾向の副産物として作成されるマニュアルは、ユーザーが本当に望み、必要としているものを供給する手助けをするツールであるどころか、製品に関する使えない専門文書であることが非常に多いからである。

●マニュアルはユーザーのツールである

大型のシステムや製品の場合、下記のように個々の利害関係や役割分担が極度に細分化されている組織では、組織全部がユーザーであるということもしばしば起こってくる。すなわち、報告書のみに関心を払う役員たち、統轄管理に役立つ手立てを求めている各部門の管理者たち、システムの維持運用を任されている上級オペレーター、システムにデータを入力したり、動作結果を確認したりする初級オペレーターや事務員、メンテナンス要員やプログラマー、監査役や品質保証検査員などがユーザーとして考えられる。

繰り返していうが、これら多様なグループに共通していえることは、システムやプログラムの開発に携わる人間にとって、これらの人々はいずれも「満足」させなくてはならない存在であるということだ。なぜなら、コンピュータに関連する商品を市場に供給している業者にとって、ユーザーは「お得意様」であり、また自分の会社組織内でシステムやアプリケーションを開発している者にとって、ユーザーは「自分の部署の上司」であるからだ。

マニュアルは、その読者がシステムから充分な利益を得ることができるよう手助けするツールである（いや、そうでなければならない）。今さらいうまでもないが、マニュアルは、コンピュータ・テクノロジーの難解さやなじみにくさを解消してくれるものであり、次のような疑問に答

えてくれるものである。「次に何をしたらよいのか？」「これはどういう意味か？」「今、何が行なわれているのか？」「どうしてこれはうまくいかないのか？」

図表 1.1　＜マニュアルの主な機能＞

機能	例
製品の使用開始を助ける	・製品の働きとその効用を紹介する ・製品の導入と使用準備をデモンストレーションする ・初歩的な操作と手順を指導する ・エラーとバグについて警告する
生産性アップと目標達成を助ける	・応用的な機能とその効用をデモンストレーションする ・生産性アップの方法を指導する ・作業の合理化を指導する ・ユーザー独自の調整および修正を助ける
問題解決を助ける	・起こり得る問題とその解決策を明示する ・専門家の助けを必要とする問題を明示する ・ユーザーが開発者に依存しなくてすむようにする

図表1.1が示すように、マニュアルは、ユーザーがその製品の使用を開始できるよう援助するだけでなく、次々に展開していくユーザーの関心に速やかに対応することを目的とし、最終的にはユーザーが開発者に依存する度合いを最小化することを目指すべきである。この図表はまた、マニュアル内の記述の多くを、どこかもっと適当な場所に移した方がよいのではないかということも示唆している。それは、オンライン・ヘルプ画面や参照画面、ビデオ・トレーニング・プログラム、リファレンスカード、ディスクに内蔵されたチュートリアル・プログラムなどである。

マニュアルのページが、バインダーに閉じられる代わりに画面上にあったとしても、よいマニュアルを書くためのこれらの基準はほとんど不変のものである。

1.2　マニュアルに関する3つの誤解

マニュアルは、文学の一形態では「ない」。マニュアルは、「単なる」リファレンスではない。ユーザー用文書が「1冊」のマニュアルのみで構成されることはごくまれである。

マニュアル作成に関するいくつかの誤った考え方に行く手を遮られると、どんなに高い理想を掲げていても、挫折することになるだろう。

●マニュアルとは文学の一形態では「ない」

マニュアルを書くということは、「英文学で最近学位を取りました」といったような、いわゆる文章家を集めて行なうべき作業ではまったくない。もちろん、たとえ科学や技術を専門とする企業でも、文学的素養を持ち、人間性豊かな者の協力によって利益を得ることはできる。確かに、マニュアルを作成するすべての企業がプロのライターあるいはエディターを少なくとも1人は必要としていることも事実である。そしてすべてのマニュアルは、言葉と文体を専門とする人間によって、少なくとも一度はチェックされなければならないということも本当である。しかし、言葉の技能とマニュアル作成の技能を混同してしまうことは愚かなことである。

この種の誤解を解消せずにおくことは危険である。なぜなら、一歩間違うと、「技術者向けの参考文献を読みこなす力のない読者のためを考えれば、マニュアルは編集前の初期の段階において簡潔で体裁のいいものに書き直されるべきであり、この書換えこそが技術文献とマニュアルとの唯一の大きな違いである」という短絡的な考えに陥ってしまうからである。

実のところ、言葉を簡明で正確にするため、どんなに技術文献のすみずみまで手直しし、文体を一新しても、ユーザーのニーズに応え切れるとは限らない。「簡潔さが足りない」「専門用語が、分かりやすい言葉に置き換えられていない」「文学的な、あるいは文体上の工夫が足りない」などの声が上がったら、それは分析が不充分なために、誤った項目立てで誤った記述が誤った筋立てでなされているからにほかならない。

●マニュアルは単なるリファレンス（参照資料）ではない

リファレンスは、マニュアルの「一部分」をなすものである。そして、この一部分はシステムをどのように操作するかということを知った「後で」、利用するものである。リファレンスとは、何を知らなければならないかを知った「後で」利用されるものを、索引形式で編纂したものである。

驚くにはあたらないが、プログラマーや開発者が「自分で作った」システムを自分で操作するときには、このリファレンスがあれば充分である。彼らはシステムがどう動くか、何をしてくれるか、どの機能をいつ使ったらよいかをよく知っているからである。プログラマーや開発者がマニュアルを必要とするのは、彼らが忘れてしまったり、最初から記憶にとどめる気のなかった用語、規則、数値データなどをちょっと“調べる”ときだけである。同様に、あるクラスのシステムを用いて、一般的な経験を積んできたユーザーは、新しいシステムとどこが異なるのか調べるだけで、新システムを運用できるはずである。

しかし、新参者のユーザーはどうか。添付のリファレンスが完璧であれば、このユーザーはシステムを操作できるのだろうか。たとえば、「フォーマット」に関する説明が、リファレンスの「FORMAT」コマンドの項目にしかなかったとしたら、どのディスクもフォーマットしなければ使えないということを初心者に伝えられるだろうか。

よくあることだが、技術者にマニュアルを書かせると、「参照」資料になってしまう。たとえ“ユーザーの知らなければならない事柄は、いちおうすべて網羅されている”としても、提示の

仕方によっては、情報をかえって不明瞭にしてしまい、結果的にマニュアルをほとんど役に立たないものにしてしまうということも事実である。

●ユーザー用文書として1冊のマニュアルだけを作成することはまれである

状況やシステムの性格次第では、必要な情報を1冊のマニュアルに詰め込むこともできる（たいてい驚異的な分量になる）が、その場合、マニュアルと他の情報伝達媒体を整理するライブラリが必要となるだろう。したがってマニュアル作成者は、1冊で充分であると確信できるとき以外は、マニュアルは2冊以上になるものと「想定」して構想を練るべきである。実際、マニュアルは数冊のセットあるいはライブラリの一部として提供されることがほとんどであるため、ユーザーが利用できるその他の関連資料の内容がどんなものかを知らずに、良質のマニュアルを設計または作成することは至難の業である。

確かにプロンプト、メッセージ、ヘルプ画面などが自然言語に近い形で表示されるソフトウェア製品がどんどん供給されるようになったとはいうものの、相変わらず1冊だけの百科事典的なガイドブックでは、いわゆるユーザーと呼ばれる読者層全体に貢献することは無理な話である。特に、ユーザー層にバリエーションがある場合、またシステムやソフトウェアが多岐にわたる機能を持っている場合はそうである。

重要なのはまさにこの最後の点である。今日では、多機能かつパワフルなソフトウェアやシステムが登場してきたため、文字通り幾千もの使用方法をそこから引き出すことができる。どのユーザーも、この数多い使用方法の中で、ごく少数の使用法にしか関心を示さない。したがって、すべてのユーザーに対して、簡潔な同じ内容のマニュアルを提供することが、意味のあることなのかという疑問が生じるのである。

もちろん、完璧な目次や索引や他の内容検索方法を設けて、マニュアルの質を向上させている企業は信頼を勝ち得ている。同様に、マニュアルの各ページを読みやすく整理している企業もよい評判を得ている。しかし、1冊の分厚いマニュアルにすべての情報を詰め込んでしまうことは、この中の情報を利用する際に、余計な負担をユーザーに与えることになってしまう。1冊のマニュアルの分量が多くなり複雑になればなるほど、読者にはそれだけ余計なエネルギーと技能が必要になる。この「全ユーザーのニーズに応える1冊のマニュアルを作成する」というコンセプトに私が反対する主な理由はここにある。つまり、有益な情報を得るためユーザーが現在負わされている負担は、少なくとも部分的にはライターが負うべきものなのである。

誤解しないで欲しい。すべての読者が使いやすいと認めるようなマニュアルを書くことができると主張しているのではない。また、ユーザーやオペレーターの一部には、マニュアルがどのように書かれているかに関係なく、これを読もうとしない人々がいるという事実を見すごしているわけでもない。私がいわんとしているのは、「使いやすい」マニュアルを作成する責任が、マニュアルの設計者と執筆者にあるのだという事実である。

1.3　優秀な人がなぜダメなマニュアルを書くのか

マニュアルを書くべきである多くの企業が、何も書いていない。書いていたとしても、ほとんどの場合充分ではなく、それを更新する努力を怠っている。そして、もっとも洗練された企業の手掛けたマニュアルでさえ、その大部分はまったく役に立たない。つまり、難解で、とっつきにくく、不明確である。

●優秀な企業は何をしているのか？

品質が高く読みやすいマニュアルをコンスタントに作成する企業は増えている。そしてこれらの企業よりも若干多い企業が、かなり高い割合で良質のマニュアルを作成している。しかし、この2つのグループを合わせても、ほんのひと握りにすぎない。

これに対して、典型的なのは**マニュアルなし**というケースである。米国内を旅行してみて、いまだに驚かされるのは、コンピュータ会社、エンジニアリング関係の会社、ソフトウェア・コンサルタント会社、銀行、メーカーなどのどれほど多くが、自社のシステムやソフトウェア製品について、マニュアルも操作に関する手引書も持っていないかということである（それどころか、**専門技術上の、あるいはシステムそのものに関する文書さえ持たない**会社のなんと多いことか）。

たび重なる要請にしぶしぶ腰を上げてマニュアルを書こうとする者がいたとしても、彼らの手でろくな結果が産み出されるはずはないのである。つまり、「急いで作って使えないマニュアル」に始まって、「金をかけたのに使えないマニュアル」に至るまでが、彼らの仕事の成果なのである。

なぜこうなってしまうのか。金銭自動出納機や、CAD/CAMシステムや、コンピュータからコピー機に指令を伝達するネットワークなどが設計でき、人材にも恵まれた優秀な会社が、分かりやすいマニュアルを作ることができないとはいったいどうしたことなのか。

これの原因は主に次の2つである。1つはマニュアル作成を軽んじる傾向があるということ、もう1つはマニュアル作成のやり方が分かっていない人間がいるということである。

コンピュータのマニュアルが出現するかなり以前から、いわゆる周辺装置に関する手引書や取扱い説明書の類はあった。この手の「作品」が主流である間はずっと、その大半がつねに読みにくいものであった。読みにくい原因は何であったのか。それは、技術と生産に携わっている人間の間に、こうした文献に時間とお金を費やすのはバカ気ていると考える風潮が昔からあるからである。古いジョークに「説明書は、その日たまたま暇だったエンジニアのうちの誰かによって書かれる」というのがあるが、もはや笑い事ではすまされない。

私の知り合いで書くのが好きな人間は皆無に等しい。私の知っているエンジニア、科学者、システム・アナリストのほとんどが書くことを嫌っている。書く作業の中で彼らがもっとも苦手とするのは、彼らよりも知識の少ない人々に、複雑な技術的概念を説明することである。

マニュアル作成について、多くの企業の間に大差がないことは明白である。これらの企業はマニュアルを作成するための時間をまともに取らず、たいてい他に“もっと急を要する仕事”を抱えている人々や、権限も力量も充分でない新入社員にマニュアル作成を任せている。しかもマニュアルの執筆と製作に関し、デザイナーやテクニカルライターなどの協力者をまったく雇わず、できる限りの費用節約を行なっている。アメリカで書かれたマニュアルはたいていの場合、熟達した編集者のチェックを受けずにユーザーに手渡される。

マニュアル作成に関心を払う企業では、若干ながら事態は改善されてきている。それでもな

お、プロのテクニカルライターも含めて、マニュアル作成者を悩ませている一番の問題は、マニュアルの書き方に関する指針や指示が充分に得られないことである。マニュアルの執筆に関係する人々の大半は、手本となる既製のマニュアルを持っておらず、ほんのひと握りの人々しかモデルとして参照できる「出来のよいマニュアル」を持っていないのが現状である。実際、アメリカにおいては、新たに雇われたマニュアル作成者のほとんどが、マニュアルなど書いたこともないという管理職やボスの下で働いているのである。

●マニュアル作成は誤解されている

マニュアルを扱いやすく、読みやすくする技術についての研究は、およそ30年の歴史を有するが、プロのテクニカルライターの多くも含めてほとんどの人が、この研究にまったく関心を示そうとしなかった。書く技術を確立することは、依然“芸術”の問題だと思われている。つまり、やや意地の悪い言い方をすれば、霊感、直感、嗜好、天性の問題ということになる。マニュアルに関する議論のあまりにも多く（特に編集やデザインに関するもの）が、個人の好みに関する論争にすり換えられている。

その結果マニュアルは、次の両極に分化することになる。一方には、マニュアル作成が嫌いで、できるだけ手間をかけず、作成されたマニュアルを評価するための基準などまったく無関心という専門技術者によって書かれるマニュアルがあり、他方には、自らの直感と文学的センスを投入してマニュアル作成にあたるが、セオリーも作成されたマニュアルや請求すべき費用の正当性を測る評価基準も持たない職人的なテクニカルライターによって書かれるマニュアルがある。

●マニュアル作成はエンジニアリングである

本書の目的は、技術面の専門家とテクニカルライターおよびマニュアル作成に関係するすべての人々に、マニュアルの有用性（マニュアルの適切さ、読みやすさ、信頼性の高さ）というものは、客観的に定義し評価することができるということを理解してもらうことであり、そしてさらにマニュアルの有用性を高めるためには、技術面の専門家とテクニカルライターの相互協力が必要であることを納得してもらうことである。**アナリストや専門技術者が単独で仕事をしたのでは、使いやすいマニュアルは作れない**。これは、彼らが文章が下手だからというわけではない。アナリストや技術面の専門家は、ほかの職業の専門家に比べて、下手な文章しか書けないわけではない。むしろ、若干の例外はあるものの、彼らはあまりに多くを知りすぎているために、彼らよりも知識の少ない読者の前でとまどってしまうのである。また、協力者に恵まれているはずのテクニカルライターも、技術面のスタッフからの“情報収集”をすべきでありながら、それをしようとしないのである（やがては、マニュアルの読み手であるユーザーが、マニュアル作成のプロセスに参入することもあろう）。

よいマニュアルを作るには、マニュアル作りの考え方を根本的に変える必要がある。有効で使いものになるマニュアルを作るのは、「デザイン」の問題ではない（同様のことが、コンピュータのプログラムにもいえる）。それは「エンジニアリング」の問題である。キーポイントは、マニュアル自体を1つの「デバイス」と考えることである。

1.4　新しい視点：デバイスとしてのマニュアル

ライターが芸術家であり、マニュアルが芸術作品であると思われている限り、プロのライターはおろか、アナリストやエンジニアでさえ、使えるマニュアルを作る見込みはないであろう。要は、マニュアルを「デバイス」とみなすことである。こう考えることによって、マニュアル作成に関するさまざまな心構えが変わってくる。たとえば、マニュアルはどのように作成されるべきか、読者から何が求められているか、マニュアルを評価する上で何を基準とすべきか、マニュアルのコストをどのように評定すべきか・・・など。

●マニュアルをデバイスとみなすと…

マニュアルや他の情報製品が芸術作品とみなされるとしたら、それらを開発するための方法を変えることは、極めて難しいだろう。これとは反対に、本としてのマニュアル、あるいはビデオテープや一連の画面などの視覚伝達媒体が、デバイス（ある特定の役割を果たす装置）とみなされるなら、マニュアルの「有用性」を向上させることができる。

マニュアルとコンピュータのプログラムとの類似性を無視することはできない。マニュアルは、プログラムがコンピュータのハードウェアに働きかけるのと同じやり方で、読者に働きかける。ただし、マニュアルの読者はハードウェアよりも圧倒的にミスを犯しやすく、記憶の信頼性においてははるかに劣る。マニュアルは、解説を行なうとともにデータを読者に伝え、読者はこれらによってシステムを正確に、そして効率よく運用するわけである。

マニュアルをデバイスとみなすといっても、マニュアルを他の名称で呼ぼうというのではない。そうではなく、マニュアルをデバイスと考えることにより、多くのテクニカルライターとアナリストが、マニュアル作成の展望を変えざるを得ないということである。この変更を実現させるためには、マニュアル作成に関するわれわれの既成概念を見直し、そうした観念にとらわれる姿勢を改める必要がある。

●作成プロセスが変わる

図表1.4にある通り、まず第一の変化は、マニュアル執筆の「プロセス」に対するわれわれの考え方に現われてくる。もしマニュアルを書くことが芸術活動であるならば、単語や文の構成や草稿作りの中に創造性があることになり、芸術家を自認する執筆者は草稿を書き、それを推

図表 1.4　<デバイスとしてのマニュアル>

	“芸術作品”としてのマニュアル	“デバイス”としてのマニュアル
最重要課題	原稿の執筆	設計のテストおよび練直し
作業手順	原稿執筆とその推敲	分析・定義→モデル作成→テスト
読者に対する評価	自主的、知謀がある	主体性に乏しい、ミスを犯しやすい
根本的価値基準	文体、アピール力、執筆者の嗜好	仕様書との合致、使いやすさ
発展的価値基準	美しさ、優美さ、“気品”	メンテナンスのしやすさ(保守性)、信頼性
出費の基準	必要ならいやいや出費	生産性、有用性
出費の方針	可能な限りの節約	利益と投資収益率(ROI)の向上

敵する作業に多くの時間をさくことになる。これとは反対に、マニュアルがデバイスであるなら、「エンジニアリング」の中に創造性があることになる。つまり、その創造性は、仕様書を書く、モデルを構築しそれをテストするといった、マニュアルの設計（あるいは原稿執筆）を実行する前になされるべきすべての作業の中にある。

●読者に対する見方が変わる

また、「読者に対する評価」も変わってくる。芸術家を自認する執筆者は、読者を自主的で明敏な人々と見るが、これでは何かを発見してそれを正確に運用するという作業が読者の肩にかかってくる。一方、マニュアルがデバイスであるならば、読者は主体性を発揮する代わりに、マニュアルの企画・設計を信頼することで、上述の負担をマニュアル作成者に肩代わりさせることができる。後に詳しく述べるが、このように読者をみるなら、ライターは、ソフトウェアがハードウェアをコントロールするのと同じように、読者の意思をコントロールすることができるのである。

●価値基準が変わる

マニュアルの「根本的な価値基準」もおのずと変わってくる。マニュアルが芸術作品であるならば、文体とアピール力が執筆の基準となる。すなわち、書いた本人には分かるが、他の人人には理解しにくい「精妙さ」とか「巧妙さ」といった感覚的な問題となる。マニュアルがデバイスならば、マニュアルが仕様書と合致しているかどうか、マニュアルがその企画・設計段階において定義付けられた役割をきちんと果たしているかどうかということが、基本的な価値基準となる。

マニュアルの価値を判断する上での「発展的な価値基準」も当然変わってくる。芸術家を自認するライターにとって、よいマニュアルとは、美しさ、優美さ、軽妙さ、“気品”を持つものである。一方、マニュアルがデバイスならば、エンジニアリングの立場から見たものが、価値基準となる。つまり、保守性（マニュアルの更新やバージョンアップがどれだけ楽にできるか）と信頼性（マニュアル通りやってうまくいくか“否か”）の問題となる。

●コスト評定が変わる

そして最後に、マニュアル作成に関する「コスト評定」も大きく変わる。芸術家を自認するライターにとって最も骨の折れる仕事は、マニュアル作成にかかる経費が妥当であるかどうかを判断することである。たいていの場合、彼らは「少なくともある種のマニュアル作成は、業務上必要不可欠なものである」という具合いに管理側を説き伏せる以外には、マニュアル作成のプロセス、および出来上がったマニュアルにかかったコストの妥当性に関して説得する術を持たない。マニュアルの“気品”や“文体”にしてもいつも説得力を発揮するわけではない。これに対して、デバイスとして作成されたマニュアルに対する評価基準は、マニュアルがさまざまな経費の節約に役立つ、すなわち利益を産み出すか否かということにある。マニュアルをデバイスととらえているマニュアル作成者は、費用の妥当性を主張することに、ほとんど不安をおぼえない。なぜなら、デバイスとは、例外なくそれにかかった費用よりも多くのものを見返りとしてもたらしてくれるからである。

1.5　マニュアルの4つの役割

今までの習慣でいえば、マニュアルは2つの大きなカテゴリーに分けられる。つまり、「インストラクション（手引書など）」と「リファレンス（参照資料など）」である。しかしこれまでの実務経験を踏まえれば、インストラクションはさらに2つのサブカテゴリーに分割されるべきであろう。つまり、「チュートリアル（初心者向け教育用）」と「デモンストレーション（上級者向け実演用）」である。さらに考察を加えるならば、第四のカテゴリーが現われてくる。すなわち、「動機付け」である。これは、ユーザーがあえてやりたがらないことを、やるように仕向ける部分である。

●マニュアルの2つの役割

よくできたマニュアルの基本単位やモジュールは、厳密にある1つの機能を果たすよう定義されている。しかし、ここでいう機能とは何か。マニュアルとは何をするものなのか。これまでのマニュアルは、以下の2つの方法でユーザーの手助けを行なうものとされてきた。

・**インストラクション（手引書）としての機能**：システムや製品がどのように作動するか、あるいは操作されるかについて教えるもの。
・**リファレンス（参照資料）としての機能**：ユーザーの記憶には残っていないと思われるキー定義、用例、コード番号などを提示するもの。

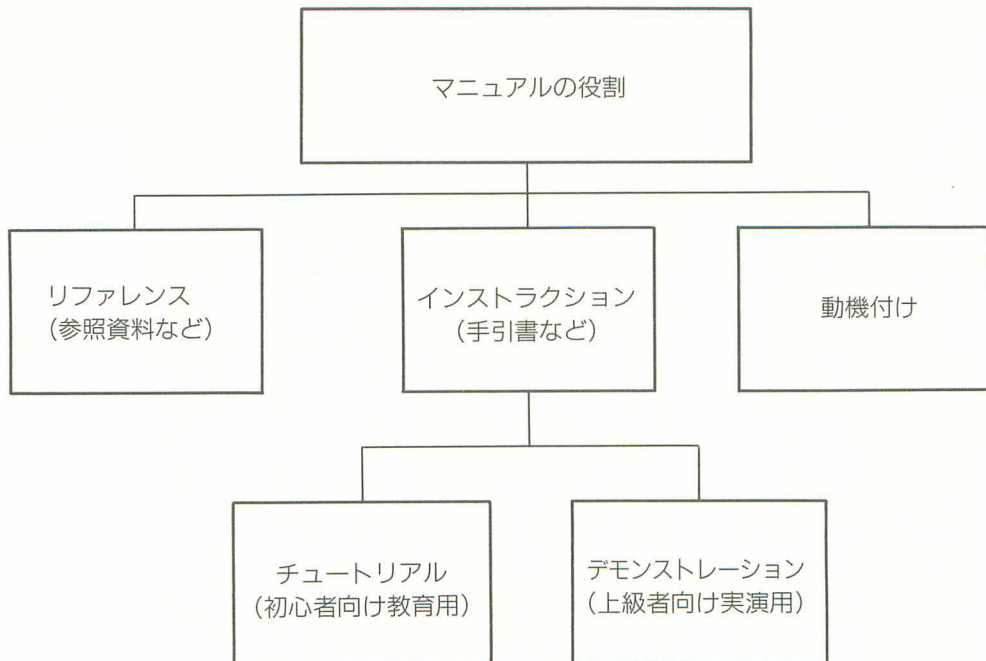
●インストラクションはさらに2つに分けられる

しかし、システムやユーザーの領域での変化にともない、マニュアルもさらに明確に細分化する必要があるように思われる。コンサルタントとしての経験から、私は「インストラクション（手引書など）」を1つのカテゴリーとして扱うのは、あまりに大ざっぱすぎると結論付けている。そこで、インストラクションを「チュートリアル（初心者向け教育用）」と「デモンストレーション（上級者向け実演用）」に分割したらどうか、というのが私の提案である。チュートリアルとは、初心者ユーザーを教育し、正しい方向に導く教材のことである。また、デモンストレーションとは、一定の水準に達している読者や、経験を積んだ読者に対し、ある仕事を行なうプロセス、あるいは作業内容を教える教材である。

「チュートリアル」は、もっとも新しい形態のマニュアルで、今日では本の体裁よりも、ディスクに書き込む形でユーザーに提供される場合が多い。この最新形態のマニュアルは、古いタイプのテクニカルライター、特にこの種のマニュアル作成に駆り出されたプログラマーや管理者にとっては、もっとも頭の痛い形式である。ところが面倒なことに、活字に対して拒否反応を示し、読むことによって何かを会得できる能力がほとんどないような読者（私は読者Xと呼ぶことにしている）が目立って増えているのである。

「デモンストレーション」とは、その名の通り、実際にやって見せて理解させるものである。このデモンストレーションは、一般的に、システムを使って何をすべきかを承知しているユーザーを対象としているため、ある業務の実行過程全体、あるいは作業内容全体が「トップダウン（総論から各論）」の形で説明されることとなる。反対にチュートリアルは、基本的な定義やコンセプトなどから説明を始めるという「ボトムアップ」の形態を取っている。

●リファレンスは経験者向け

プログラマーの中には、「リファレンス」を、誤ってマニュアルそのものと同一視する者が

図表 1.5　＜マニュアルの4つの役割＞

いるが、リファレンスとは、実例や情報を要約して提示するものであり、何を知りたいかが分かっている人にのみ有効なものである。経験を積み、高いレベルに達したオペレーターやユーザーは、このリファレンスのほかに何も必要としないであろうが、初心者および高いレベルに到達していないユーザーやオペレーターは、リファレンスだけでは不充分である。

●4つ目の役割：動機付け

ユーザーの全体像の変化により、マニュアルには以上3つの機能のほかに、「動機付け」という4番目の機能を付け加える必要が出てきた。つまり、ユーザーがあえてやりたがらないことを、やるように仕向けるためのマニュアルの機能である。動機付けとは、要するに発想と手法の売込みである。すべてのマニュアルがこの動機付けを必要としているわけではないが、現状では動機付けがあまりにも欠如している。簡単にいえば、システム上の問題の中には、ユーザーが見すごしていることが原因というよりは、ユーザーの消極的対応を解消できないという点で、開発者側が責めを負うべきものが、数多く存在しているのである。「消極的」だったり、「欲張りすぎ」だったり、あるいは単に「怠慢」だったりすることが原因で、システムは多くの場合、われわれの考える本来の方法で使われてはいない。

あたりまえのことであるが、ユーザーの動機付けを行なうには、ユーザー教育のためのマニュアルとは種類の異なる書き方、つまり異なる形のユーザー・コントロールを行なわなければならない。これは、新米の事務員と経験豊かなインストーラーに対する教育が異なった方法で行なわれるのと同じくらい、自明なことである。

第2章

要件：マニュアルの成功と失敗を分けるもの

2.1　マニュアルの最終目標：コントロール

効果的なマニュアルを書こうとすると、次のようなパラドクスにぶつかる。「こちらの意図をうまく伝えるためには、読者の主体性や理解力を尊重しなければならない。しかし、読者をかいかぶりすぎてはいけない。また、ほとんどの読者は、一方的な押しつけや命令には憤慨するのだが、押しつけや命令を必要とする読者は非常に多い」。ユーザー・マニュアルやオペレーション・マニュアルを書く場合の最良の戦術とは、こうした一般的な読者のもつ弱点とうまく付き合うことであり、いいたいことの伝え方を自在に「コントロール」することである。

●読者を情報処理のシステムとみなすべし

マニュアルが、種々雑多な情報の寄集め以上の何ものでもないとしたら、マニュアルの有用性は初めから読者の理解力と懐の深さの問題ということになってしまう。反対に読者の能力と器量に合わせてマニュアルが作成されているならば、読者がマニュアルを誤って使用しないよう、ある程度対処することができる。

読者の能力と器量に合わせたマニュアル作りということを説明するのに、「コントロール」という言葉を使うと、そうとう非難を浴びそうである。人間をコントロールするということになれば、深刻な倫理的問題を引き起こす。

断わっておくが、私はマニュアルを書く際に、読者に何かを強要するような書き方をすべきだといっているのではない。また、たとえテキストや図表などの一部であっても読者の理解を強要できるなどとは考えていない。むしろ、効果的なマニュアルを書きたいと思っている人には、読者というものを複雑な情報処理のシステムと見なして、**そのシステムの引き起こすノイズやエラーの原因となるものをコントロールする術**を学ぶべきであると提唱したい。

人間の読解力や理解力については、未知の部分が多い。しかし、まったく分かっていないわけではない。われわれは、読者を理解不全に陥らせたり、読者に間違いを犯させる原因について多くを知っているし、記憶力の効用についてもある程度分かっている。少なくともこれだけはいえる。マニュアル作成者は、マニュアルの正しい使用を妨げる原因となり、特に読者の誤用を誘発すると思われるものを、マニュアルから排除しなければならないと。

●マニュアルは読者をコントロールする

要するに私は、マニュアルや他の情報製品は、コンピュータのプログラムがコンピュータに働きかけるのと同じように、読者に働きかけるということがいいたいのである。つまり、マニュアルもプログラムも、その対象となるものをコントロールする。そして、設計不良のプログラムが、システムの故障、中断、貴重な資源の無駄使いの原因となるように、マニュアルも設計をきちんとしないと、読者の理解不全、誤り、さらには作業の中断を引き起こしかねない。また、テスト段階のプログラムが、実行するたびにさまざまなバグを発生させるように、マニュアルもテストを怠ると、読者がそれを手にするたびに誤りと矛盾をさらけ出してしまう。

読者やユーザーをコントロールすることは、結局は読者やユーザー自身の利益につながる。その狙いは、読者の意欲をそぐことも、作業能率を引き下げることもなく、読者の求めているものを、もっとも効果的な筋道で読者自身が見出せるような仕組みにマニュアルを設計することである。

（私はすべての文筆業務について－ビジネス関係の文書作成の場合でもそのすべてについて－上述の考えを推奨するわけではない。文学というものは、読者の想像力、経験、知的能力に訴えかけることで成立している。卓越したエッセイを理解するには、精読と考察が多くの場

合に必要となる。しかし、内容を理解するために考察を必要とするようなマニュアルは、役に立たないのが常である。）

辞書	操作手引書
用語解説書	操作手順説明書
目録	企画書
登録簿	提案書

低 ……………………………… コントロールの度合い ……………………………… 高

図表 2.1　＜マニュアルにおけるユーザー・コントロールの度合い＞

●コントロールの2つの極

すべての出版物は、図表2.1の線上のどこかに必ず位置付けられる。**もっともよくコントロールされた文書とは、読落とし、読飛ばし、拾読みがされることなく、最初から最後まで通して読まれるものである。**このグループの中でよく知られているものには、システム導入計画書(installation plan)、システム組成手引書(assembly instruction)、初期トレーニング・プログラム(introductory training program)、提案書(proposal)、および仕様書などがある。この種のカテゴリーに含まれる文書は、ほぼ次の3つのタイプに分類できる。

- **段階発展型**（次第に複雑な内容へと段階的に説明を積み重ねていくもの）
- **連繋手順型**（つながりを持つ一連のステップや作業を提示するもの）
- **論理展開型**（主張を帰納的あるいは演繹的に展開するもの）

この対極にあるのが、筋道立てて読まれることのない文書である。辞書、用語解説書、目録、登録簿というような、参照されるべき内容がアルファベット順や番号順に並べられたものがこれである。もっとも、このようなコントロールの度合いが低い文書でも、読者をコントロールすることには意味がある。巧みに設計されたリファレンスにおいては、読飛ばしや回り道をすることもなく、“1回の検索”で必要な情報にたどりつくことができ、読者は迅速に目的の情報をそこから引き出すことができるからである。

2.2 マニュアルを評価する4つの基準

マニュアル作成を、勘にたよるやり方から、文書作成工学に転換するための第一ステップは、どんなマニュアルが望ましいかをテストする（あるいはマニュアルの設計をテストする）ための基準をはっきりと定義することである。たとえば、ある会社がその基準を受け入れることができるなら、その会社は出版物の品質を評価するための指針あるいは尺度を、それらの基準に照らして独自に開発することも可能なはずである。出版物を評価する際にもっとも役に立つ基準を、レベルの低い方から高い方へ順に挙げると、「可用性」「適合性」「アクセス容易性」「可読性」ということになる。

図表2.2 は、品質保全工学の考え方を、ほんの少しマニュアルの草稿作成（あるいはヘルプ画面やチュートリアルの作成）に応用したものである。明らかにマニュアル作成には、品質評価に関して、少なくとも4つの基準がある。これは、段階を踏んで徐々にレベルが高くなっていくものであり、その段階は「可用性」（これがなくてはどうにもならない）から始まり、「可読性」（言葉として明快で理解しやすいこと）で終わる。

●可用性（ユーザーが利用できること）

いまだに、ユーザー用文書を作成しようとしない開発者がいる。プログラマーだけで構成されている情報処理（DP）会社の場合は、たいていそうである。このような会社は、「ユーザーは何をするのか」「ユーザーは何を知っているのか」「ユーザーはどのように作業をするのか」といった「ユーザー側の事情」にまったく無頓着である。したがってこのような会社は、ユーザーの事情に通じている人を雇い入れない限り、いつまでもマニュアル作成の重要性に気付かないことになる。

●適合性（ユーザーのニーズに合致していること）

今日では、販売者および開発者の大半が、少なくとも何らかのマニュアルを作成している。しかし残念なことに、彼らはマニュアル作りに対し、百科事典でも編纂するようなやり方で臨む傾向にある。つまり、すべての情報を恐るべきボリュームの1冊にまとめようとし、ユーザーに自分で必要な情報を探させようとする。彼らがしなければならないのは、どのようなマニ

基準	必要作業
可読性 （スラスラと簡単に読めるか）	専門家による編集
アクセス容易性 （構成が適切か）	企画、設計
適合性 （ユーザーの業務と使用目的に合致しているか）	分析
可用性 （必要なときに利用できるか）	ユーザー事情への精通

図表 2.2 ＜マニュアルの品質基準とそのレベル＞

ュアルが必要とされるのかを「分析」する作業である。つまり、ある特定の読者の業務と使用目的にぴったり合うマニュアル作りが必要なのである。特定の読者のニーズに合わせることができなければ、“分かりやすく”作ったはずのマニュアルも、不適切で、使いにくく、信頼性に欠けるものになりかねない。

●アクセス容易性（ユーザーが情報をアクセスしやすいこと）

ユーザーのニーズに応えるものをちゃんと含んでいても、マニュアルの構成が使用に耐えないほど入り組んでいる場合がある。このような場合読者は、ページを飛ばしたり（スキップ）、わき道にそれたり（ブランチ）、同じところを循環したり（ループ）、回り道（迂回）をしたりしたあげく、結局わけが分からなくなってしまう。ソフトウェア・エンジニアリングの用語を借りれば、あまりにGOTO*の多いこのようなマニュアルは、信頼性に欠けるものである。このようなマニュアルは、読み進む筋道が非常に入り組んでいるため、どんなに有能な読者でも、おそらくどこをどう読んだらいいのか分からなくなってしまうだろう。筋道を考えてマニュアルを「設計」し、その設計をテストして、バグを取り除く作業を怠らない企業だけが、スムーズに読むことのできる、GOTOのないマニュアルを作成できるのである。正確で筋道の一貫したマニュアルこそが、「業務別マニュアル」と呼ばれるのにふさわしい。これは、ユーザーの業務は何か、システムや製品をユーザーがどのように使用するか、ユーザーがどんな情報を必要としているかについて、開発者がきちんと分析しているマニュアルを指す。つまり、「業務別マニュアル」とは、ユーザーの求めるものだけが分かりやすい筋道で構成されているものである。

●可読性（文章が読みやすいこと）

マニュアルが正確で筋の通ったものでも、マニュアルの最終的な出来ばえは、その読みやすさにかかっている。つまり、マニュアルの対象読者が、いかに容易に、また正確にマニュアルを理解することができるかという問題である。情報処理（DP）のかなり多くのプロが、いまだに言葉と文体の問題を“飾り”程度に思っている。プロのライターやエディターによる簡単なチェックさえ受けていないマニュアルやインストラクションが山ほどあるのは、そのためである。「専門的な編集作業」を経なければ、最高品質のマニュアルはあり得ない。

したがって、ユーザーが求める高品質のマニュアルは、「ユーザー側の事情への精通」、必要な情報の「分析」、「設計」と「テスト」、「プロの手による編集」によって初めて達成されるのである。もちろん、これらの作業がどの程度までやれるかは、個々の事情に制約される。しかし、各作業の最終期限を守りながら、利用できる機会をなるべく活かして各作業の内容を充実させるべきである。

訳注＊ GOTO：プログラミング言語の命令文の1つ。本書では、構造化プログラミングの手法を借りて、マニュアル作成を構造化（モジュール化）した方法論が語られるが、GOTOとは、モジュールの一連の正規の流れからいったん外に出て、まったく異種の処理を行なうため、他のモジュール「の方へ行く」（go to）こと。

2.3 犯しやすい3つの誤り

4つの評価基準に照らして高品質といえるマニュアルを作るためには、まず正しい企画・設計がなされていなければならない。もしマニュアルがこの基準のどれかに照らしてみて、低い評価しか得られなかったら、「方針上の失敗」、「構造上の欠陥」、「表現上の失策」のうちのどれかに、その原因を見出すことができるはずである。

マニュアルの欠陥の多くは、初稿が執筆される前のミスに起因する。しかし、この点はほとんど指摘されない。また、マニュアル作成におけるもっとも深刻な欠点は、第一稿が完璧に出来上がってしまった後では、ほとんど修正が不可能であるということだ。しかし、この点もほとんど指摘されない。

マニュアルの品質を損ねる誤りは、おおむね3つの段階で発生する。そして、編集の段階で修正できるのは、その中のわずか1つである。

●方針上の失敗

まず最初の段階で犯される誤りは、「方針上の失敗」である。これは、プランニングと分析における失敗である。つまり、どんなマニュアルがいかなる読者に求められており、その読者はどんな作業に携わっているのかについて、正確な定義が行なわれていないために発生する失敗である。主な「方針上の失敗」は以下の通りである。

・どんなマニュアルが必要とされているかについて、プランニングと分析を行なう必要性を見すごしている。
・製品やシステムには適合しているが、ユーザーの使用目的や作業内容を反映していない。
・1冊の汎用型マニュアルにしなければならないと考えてしまう。
・マニュアルを対象読者のボキャブラリや読解力に合わせない。

●構造上の欠陥

「構造上の欠陥」とは、マニュアルの企画・設計とモデル構築における失敗である。つまり、マニュアルのアウトライン作成が充分に行なわれていない、アウトラインに対するチェックがあまい、細部の執筆を開始する前にマニュアル作成のプランに関してテストを行なっていない、などである。たとえ方針上の失敗がマニュアルに1つもなくとも、構造上の欠陥があればマニュアルの「正確さ」は低下する。そして、さらに重要なことは、構造上の欠陥がマニュアルの筋道を混乱させ、極めて使いにくいマニュアルにしてしまうことである。もっとも一般的な構造上の欠陥は以下の通りである。

・マニュアルのアウトライン作成もほとんど、あるいはまったく行なわず、その他のマニュアル仕様書も作成しない。
・レベルの低い見せかけのアウトライン作成手法にたよってしまう。
・アウトラインと仕様書に対して厳しいチェックを行なわない。
・対象ユーザーと読者を設計の段階から排除してしまう。

●表現上の失策

「表現上の失策」とは、編集と校正における失敗である。つまり、名称の不統一、文法上お

よび表記（スペル）上のケアレスミス、ぎこちない未編集の文体、意味の曖昧な文章などである。表現上の誤りは、「初稿を編集するにあたり、どうすれば内容を明確に伝えられるか」について充分心得ている人間が社内にいない場合、あるいはただ単に会社がエディターに作業に必要な時間を充分与えない場合に発生する。

しかし、編集段階にはパラドクスがあることに注意して欲しい。まず、能力ある文章の専門家によってチェックを受けずにマニュアルを印刷してしまうことは重大なミスである。また、書き方の悪いマニュアルが、エディターの技能によってよくなると信じるのはもっと危険である。

したがって、マニュアルを使いやすくするには、以下の3つのテストを通さなければならない。

・どんなマニュアルにすべきかという方針はよいか、そのマニュアルが特定のユーザーやそのニーズに合致しているか、情報製品のセットまたはグループの中でそのマニュアルが過不足なく適切な役割を果たしているか、について検証する方針上のテスト。
・マニュアルの構成要素がアクセスしやすく、信頼できる筋道の中に置かれているかについて確認する構造上のテスト。
・文章や図表にケアレスミスやぎこちない感じがないか、加えて、どこにどう続くか分からないような曖昧な文の流れで読者を悩ませていないか、について確認する表現上のテスト。

図表 2.3　＜犯しやすい3つの誤り＞

誤り	原因
方針上の失敗 （プランニング/分析の失敗）	・対象読者の明確化不足 ・ユーザーの作業内容の明確化不足 ・全般的計画の欠如
構造上の欠陥 （設計とモデル構築の失敗）	・独立アウトラインの欠如 ・アウトラインに対するテスト不足 ・検討作業からのユーザーの排除
表現上の失策 （編集と校正の失敗）	・ケアレスミス ・文体のぎこちなさ ・未熟な編集

2.4　マニュアルの品質向上を妨げるもの

他に選ぶべき方法を示された後なお、なぜ多くの会社が、質の低い、欠陥だらけのマニュアルを作り続けようとするのか。それは品質の向上に逆らって働く、組織的かつ心理的な強い力があるからである。すなわち「論争に対する恐れ」「近視眼的発想」「性急さ」「完全主義」の4つの障害である。

多くの会社と多くのライターが、本書の示す方法に、心の底から賛同している。しかしそれにもかかわらず、彼らはこの方法を現実に実行していない。彼らは、本書の問題設定と評価基準を支持している。また、本書がこれから述べる方法によって、問題が解決され、高い品質のマニュアルが作成できる、という点にも同意している。しかし、彼らはいつまでたっても、この方法を実行に移せないのである（ソフトウェアの品質コンサルタントも、同じような問題を報告している)。この理由は何なのか。

明らかにもっとも利益の上がることが、実行に移されないということは、この利益に対抗する他の利益が存在しているからだとみてほぼ間違いない。マニュアル作成の場合、品質の高いマニュアルの作成を妨げ、同時に質の高いシステムの開発をもしりぞけてしまうこのような一連の「障害」は、人々を最善の方法から遠ざけ、より安易な方法である“早かろう悪かろう”式の方法を選ばせてしまう。これらの障害を図表2.4に掲げる。

図表 2.4　<マニュアルの品質向上を妨げるもの>

障害	傾向	結果
論争に対する恐れ	意見対立と論争からの逃避	テスト実施と問題解決の遅れ
近視眼的発想	短期コストに対する執着	長期的利益の損失
性急さ	独善的な最終期限への固執	製品/マニュアルの開発不全
完全主義	推測と予測に対する嫌悪	優柔不断、無意味な遅れ

●「論争に対する恐れ」

これは、方針や優先課題の決定に関するすべての対立を回避しようとすることである。特に、課や部の間における意見対立の回避などにみられる。これは、理解をなかなか示してくれない人、あまりにも多くの質問をする人、反対意見を提示する人などに対して、意思の疎通を図ろうとしないことである。官民を問わず、大規模な官僚主義がはびこるところでは、論争の回避が重大な目標となり、利益や有用性や品質の追求といった事柄よりも優先されるのである。

●「近視眼的発想」

これは、現時点において見通せる範囲内(進行中のプロジェクト)、または計算できる射程内(決算期）にあるもの以外に対しては、コストも成果もいっさい考えないことである。これは、将来に向けてのコストと利益を見積ろうとしないこと、すなわち短期的コストに重きを置き、長期的利益にまったく目を向けないことである。たいていの企業の出版部門は、この近視眼的な発想にとらわれており、その結果、出費を上回る利益が予想される事柄でも実行に移すことができずにいる。

●「性急さ」

これは、たいていが（あるいはほとんどが）変更の可能な事柄を、あらかじめ決められた通り行なわなくては気がすまないことである。この症状は、近視眼的発想とも似ているが、特に急いで事を運ぶことを強調し、出来上がりの表面的な判断基準(ページ数、プログラムのステップ数など）を絶対化しようとする。事態を一層複雑にするのは、口では品質重視というが、実際には単に性急である多くの管理職の態度である。タイミングのよい“早かろう悪かろう型”の仕事をほめ、遅れた仕事は無条件に非難するのである。

●「完全主義」

これは、プランニングや価値判断に対して不平を述べることであり、はずれるかもしれない見通し、あるいは完了していない仕事や、やりかけの仕事についての討議をしぶることである。実務の中での完全主義は、優柔不断の姿勢とほとんど変わらない場合が多い。不充分なデータにもとづく判断を避け、関わらないようにすることは、何の決断も行なわないのと同じである。しかも、追加情報を与えると、完全主義者は多くの場合、一層混乱してしまうのである。

このような障害に進路をはばまれると、やる気のあるマニュアル作成者は、間違いなく欲求不満に陥る。このような作成者に投げかけられるのは次のような言葉である。“この問題は、後回しにしよう”（論争に対する恐れ)、“色刷りのコストが高すぎる”（近視眼的発想)、“とにかく金曜日までに終わらせてくれ”（性急さ)、“まず初稿を見た後で、どんな図表を使うべきかを決めよう”（完全主義)。

第3章

“有用性”：システムとしてのマニュアル

3.1 “玄人向け”から“素人向け”へ

技術者や発明家が、新しい製品や新しい技術の開発に真剣に取り組むときには、その製品が使いやすいかどうかには、ほとんどこだわらないのが普通である。しかしながら、1980年代に入ると、「有用性」が、コンピュータのソフトを企画する人間にとってもっとも重要な課題の1つになった。

●システムの評価基準の推移

今日のコンピュータ・システムの大部分は、1950年代のコンピュータが行なっていた作業と同じ作業を行なっている。異なる点は、スピードが速くなった、安くなった、人間の行なう手間が少なくなった、ということである。情報処理産業が登場して以来、コンピュータに対する評価基準は変わり、コンピュータにつぎ込まれる費用も、時を追うごとに急増してきている。

図表3.1に見る通り、システムの品質に対する1950年代の一般的な基準は、「とにかくシステムが作動するかどうか」という点にあった。年月が経つにつれて、アナリストとエンジニアの関心は、「経済効率」へと向けられていった。つまり、スループット(一定時間あたりの仕事量)、サイクル・タイム（仕事あたりの所要時間)、資源（仕事に必要となる人材、機材、資材など）である。現在のものよりはるかに処理速度の遅いCPU（中央処理装置）のリース料のために、1秒あたり60ドル支払わなくてはならなかった時代においては、経済効率が重要だったのである。

しかし、コンピュータとメモリーの価格が低下するにつれて、経済効率は、多くのところであまり尊重されなくなった。現在では、マシンそのものの効率をアップするよりも、コンピュータをいかに効率よく使用するかを考える方が、個人にとっては安くつく場合が多い。今日においてもっとも重要で、よく話題に上がる実務的な基準は、「メンテナンスのしやすさ」である。つまり、システムの修正や調整や機能強化（バージョンアップ）などがいかにしやすいかということである。

1980年代に入ると、システムのテーマはいく分変わってきた。しかし情報処理（DP）会社や経営情報システム（MIS）会社の多くは、依然として1970年代のレベルにさえ到達していない。

基準	年代
信頼性と有用性 (中断がないこと)	1980年代
メンテナンスと変更の容易さ (保守と機能強化が簡単であること)	1970年代
効率のよさ (コンピュータの性能を充分引き出せること)	1960年代
何らかの成果が得られること (とにかくシステムが作動すること)	1950年代

図表 3.1　<システムに対する品質評価基準の推移>

というのは、彼らは新しい開発手法のメリットを活用せずに、メンテナンスのしにくいシステムを組み上げ続けているからである。ところが、最近のテーマは“ユーザーにとってのなじみやすさ”、つまりシステムをいかに使いやすくするか、ということである。

コンピュータ・テクノロジーは、1つの開発サイクル全体を完結させた感がある。ほとんどの部分において、やっている内容は初期のころと変わりはない。変わったのは、ユーザーに対する姿勢が親切になったという点である。今日のオペレーターは、数学者やプログラマーではなく、事務員やビジネスマンが主流であり、10歳の子供さえいる。エンジニアは、今日の新しいシステムを“専門家向け”と考えてはならない。そうしたいのなら、ユーザーが専門家によって占められていたコンピュータの草創期にまで、さかのぼらなくてはならないだろう。

●システムの有用性とは何か

“ユーザーにとってのなじみやすさ”というよりも「有用性（使いやすさ）」といった方が分かりやすいだろう。この有用性とは、使いにくさを「巧みな設計によって解消すること」である。つまり、デバイス、システム、プログラムなどが持つ表面には現われない部分の操作性によって、それら自身が可能な限り使いやすい状態になる、ということである。どのようにマニュアルが書かれるにせよ、20回キーを押さなければならない作業よりは、2回の方が楽である。また、“画面クリア”のボタンが“文字挿入”のボタンのすぐ横に置かれていたら、マニュアルでいくら注意を与えておいても、うっかりして画面を消去してしまう場合が出てくる。しかし、ボタンの位置をほんの少し移動させれば、この誤操作の頻度は低下するのである。

もちろん、有用性と対立するさまざまな概念がある。たとえば、初心者が操作を覚えやすいようにシステムを設計したとしても、長期的に見て、その設計が日々の操作を簡便にするとは限らない。有用性とは、システムが充分に企画・設計、テストされていなければ得られない結果である。有用性は、充分な分析と企画・設計によって、初めて生まれるものであって、いくら立派なマニュアルを書いてもシステムが使いやすくなるわけではない。

特にマニュアル作成者にとって、「有用性」という言葉は、2つの重要な意味を持つ。第一は、システムが操作される上での使いやすさであり、第二は、マニュアルが使用される上での使いやすさである。言い換えるならば、本書を通じて私が主張しているように、マニュアルを1つのシステムと考えると、**コンピュータ・システムの有用性が充分発揮されるか否かは、マニュアルの有用性のレベルによって決まる**ことになる。マニュアルが使いやすければ、コンピュータ・システムが、その本来の使いにくさよりも、使いにくくなることはない。マニュアルと他の情報製品が、ともに最高の有用性を持っているならば、これらで説明されたシステムは、意図された通りに使いやすいものとなるだろう。しかし、マニュアルに誤りや欠陥があればあるほど、システムはその有用性を低下させていく。

3.2　マニュアルの第一原理

システムやソフトがそれぞれ固有の有用性を持っているように、システムに添付されたマニュアルにも、それ自体の有用性がある。しかし、どんなに出来のいいマニュアルでも、システム自体の持つ欠点を充分に補うことはできない。そこで第一原理はこうである：
完全無欠のマニュアルが、出来の悪いシステムを改善することはあり得ない。

●システムの不備はマニュアルでカバーできない

ひとたびシステムが動き始めると、もはやその操作性を根本的に変えることは、ほとんど不可能になる。仮に改善されたとしても、ほとんどの場合それは表面的なものにすぎない。システムの不備を“とびきり出来のいい”マニュアルでカバーしようとするような方法で、既存のシステムの有用性を高めようとしても、まったく効果は期待できない。(要するに、道にあいている穴をそのままにしておいても、警告さえすれば危険ではない、とたかをくくるようなものだ。)

地図を描いたからといって、ジャングルを緑園に変えることは不可能だ。同様に、面倒でややこしい処理手順は、出来のよいオペレーション・ガイドを作っても、ユーザーにとって親しみやすくはならない。事後にマニュアルが作成される場合（最初にシステムを開発して、その後にマニュアルを作成する場合)、そのマニュアルでシステムの分析、企画、プログラミングなどの欠陥を補うのは、絶対に無理である。

本書は、マニュアルに関するものであるはずなのに、このように書くのは蛇足だと思われるであろう。しかし、マニュアルに過度な期待を寄せるのは間違っている。マニュアルで設計ミスやプログラムのミスを正そうとしてはならないのだ。

処理手順の簡単なシステムは、きちんと説明されれば、とても分かりやすいものである。逆に難解な処理手順のものや、あぶなっかしくてトラブルを起こしやすいものは、いくら説明がよくても、依然としてそのような欠点が改善されるわけではない。出来の悪いマニュアルが操作手順を分かりにくくすることはあるとしても、優れたマニュアルが手順を簡単にすることはあり「得ない」。

●マニュアルはシステムが完成する前に書け

マニュアルがシステムを改善する方法はただ1つ、システムが出来上がる前に、マニュアルを作成することだ。マニュアルのライターは、システムの“最初のユーザー”となるわけだから、システム開発者が見落としがちな改善の方法を見つけることができる。そして、システムがディスクに組み込まれる前に、マニュアルをきちんと書くことができれば、システムを設計し直す時間もできるのではないか。

図表3.2で分かるように、マニュアルがシステムの機能仕様書作成の段階で書かれる(少なくとも設計される）ならば、そのマニュアルは、システム設計のモデルとして使用できる。開発者はシステムが人間と直接関わる部分ーいわゆる“ユーザー・インターフェイス”ーのエラーや曖昧な部分を見つけ出し、訂正することができる。システム設計の段階に入ってからでも、説明しにくい操作手順が見つかれば、文書化を進めているシステムの各モジュールを手直しするチャンスは残っている。この種の手直しは、この段階ならまだ可能である。しかし開発の最終段階になると、マニュアル作成者は多かれ少なかれ、システムの仕様に制約されてしまうものである。

要するに、有能なライターによる分かりやすい解説をはねつけるような操作手順は、たいて

図表 3.2 <マニュアルの第一原理>

マニュアルの開発時期	マニュアルの働き
製品/システムの機能仕様定義時	・製品の操作手順と開発方針が明確化できる ・ユーザーに不親切な要素が識別できる ・ユーザーの目標達成率を高められる
製品の設計/プログラミング時	・バグやエラーをつきとめられる ・製品不信の原因を明らかにできる ・設計者の迅速な決定を促すことができる
製品の引渡しおよび使用時	・製品にユーザーが適応できるようにすることができる ・システム内のバグに対して警告を発することができる ・製品に対するマニュアルの責任は回避される

い理解しにくい。電車の時刻表ほどもあるような例外集と注意書きがなければ説明できないような業務は、おそらく能率よく行なわれることはないだろう。適切な環境があれば、システムあるいは操作手順が確定する前、すなわちシステム変更に費用がかかり、しかも面倒になる前に、これらの問題点がはっきりするはずだ。マニュアル作成者が、操作手順の変更を促すことによって、回りくどい説明を要したページが、数ページまるごと削除できることもある。雑然として曖昧な操作手順ほど、マニュアル作成をやっかいにするものはない。逆に、マニュアルの記述が雑然として曖昧であれば、使いやすいシステムも使いにくくなってしまう。

皮肉なことだが、次のことがいえる。システムの設計がよければよいほど、マニュアルの必要性は低くなる。システムの不備を早い時期に見つければ見つけるほど、書くべき量を減らすことができる。

3.3　マニュアルの有用性：その定義と評価

マニュアル作成の課題が有用性を考慮していかに企画・設計するかということであるなら、そしてその作成過程が“芸術”とは無関係のものであるなら、そこには何かしら一定の評価基準が必要である。有用性のための指標としては、まず次のことを考えるべきである。対象となる読者が部分的に読み飛ばしたり、読み返さなければならないような事態が増えれば増えるほど、そのマニュアルは使いにくくなる。

●スキップ（読飛ばし）や後戻りはマニュアルをダメにする

マニュアルの出来がよいからといって、必ずしもシステムが優れているとはいえない。しかし、システムを成功させるためには、出来のよいマニュアルが不可欠な要件となる。

評価の基準－有用性を高めるための指標－はいろいろ考えられるが、**マニュアルを書く前、**すなわち予想される不備や誤りが全部手直しできる時点で使えるような評価基準を設定することができるはずである。「製品は開発期間の後になればなるほど変更に費用がかかり、したがって、実質的に変更不可能になっていくものである」。この基本を忘れてはならない。

経験に照らし合わせて、私は次のように結論付ける。対象となる読者が**マニュアルを使う際、何度スキップしたか、何度後戻りしたか、その回数**がもっとも適切な有用性の指標となる（もちろんこれは「逆説的な」言い方である）。

だからといって、何もすべてのマニュアルについて、あらゆる読者が最初の一語から最後の一語まで、順番に読めるように作成すべきだというのではない。実際、そのようなマニュアルは皆無といってよい。それでもスキップや後戻りは、わざとにせよ、そうでないにせよ、マニュアルを使いにくくするといいたい。

指標の定義の中で“対象となる読者”という言葉を使ったことに注意していただきたい。読み手の目的や状況が異なれば、当然、同じマニュアルでも使い方は違ってくる。すべての文章を順番に読む人もいるだろうし、飛ばしながら拾い読みをする人もいるだろう。また同じ読者でも、1度か2度通して読んだ後、飛ばしながら拾い読みをすることもある。当然、マニュアルの読者層が多様になればなるほど、あらゆる読者に使いやすいようにするのは困難になるものだ。

●3つの誤りに照らしてみると

おもしろいことに、スキップ（読飛ばし）やループ（循環）（分岐、迂回、GOTO）は、マニュアルの3種類の主な誤りに対応させることによって、3つのグループに分類できる。

- 「方針上の失敗」は、もっとも大きいスパンのスキップや方向転換の原因となる。
  本が読者の目的に合っていなければ、読者は必要なものが見つかるかそれを諦めるまで、本から本へ飛び移ることになる。1つの仕事に2冊の本が必要になるということは、ユーザーのニーズや利益が、本の内容選択や分割の仕方に反映されていないということになる。また、ユーザーがあるマニュアルの98%を無視しなければならないとしたら、そのマニュアルは別のユーザー向きに設計されたもの、ということになる。
- 「構造上の欠陥」は、中間的なスパンのループ（循環）やスキップ（読飛ばし）の原因となる。この種のマニュアルには、前から後ろへ飛ばさなくてはならないことがたびたびあるが、他のページにある図や表、一覧などを参照するよう指示してある場合は、特にその回数が多くなる。（注：**マニュアルの有用性を妨げる一番大きな原因は、本文**

図表 3.3　＜マニュアルの有用性を評価する＞

基準	誤り	結果
可用性 適合性 アクセス容易性 可読性	方針上の失敗	・本から本へ飛び移る ・1つの作業に2冊の本を必要とする ・大半のページを無視しなければならない
	構造上の欠陥	・本の最初から最後に飛ぶ ・ページ通りに読まない ・図や表を探し回る
	表現上の失策	・ケアレスミスが気になって集中できない ・不統一な用語に当惑する ・難しい部分を読み返す

と参照すべき図表が離れているということである。）

・「表現上の失策」によるGOTOは、一番小さいスパンのものである。普通は1段落か1ページですむ。編集がまずいために、読者は意味が不明瞭な文や、専門用語の不統一性や、文法上の誤りなどにとまどってしまうに違いない。有用性を損なう原因としては最も小さいものだが、このようなミスがあると、構成のしっかりした優れた本でも、評価は低くなる。

以上のように定義しておけば、有用性を企画・設計の初期の段階で評価することができる。方針上の失敗は－プランをテストする場合に－マニュアルのプランの中にループ（循環）を発生させる原因となる。構造上の欠陥は－アウトラインを正確に組み立て、テストする場合に－そのマニュアルのアウトラインの中にループやGOTOを生じさせる。表現上の失策だけは、マニュアルの草稿が完成してから現われる。

3.4　よいマニュアルは並列型ではなく直列型

並列型のマニュアルとは、ソフトやシステムの操作に関するあらゆることを、説明の対象となるものの性質ごとに構成したものである。直列型のマニュアルとは、ある一定の事柄を、読み手が実際に操作する手順や業務内容に沿って構成したものである。並列型のマニュアルは、「リファレンス・ブック（参照書）」である。したがって、たいていの場合、ユーザー教育用、あるいはデモンストレーション用としては使いづらい。

●どんなに多機能でも目的は１つ

コンピュータの出初めのころは、プログラムの機能は1つか2つしかなかった。選択肢、あるいは“ユーザー定義オプション”は、ほとんどなかった。したがって、Run-Book（実行指示書）*は単純で、すっきりしていて、読みやすく、作業本位にできていた。

しかし、今日のシステムの傾向は、できるだけ多くの適応業務を遂行する方向に向かっている。そして高級言語、データベース管理システム、あらゆる統計分析やグラフィック表示を行なうパッケージなどがそのために作られている。

このような多目的システムのユーザー・マニュアルやオペレーション・マニュアルを作成する場合、重大な方針上の問題が生じる可能性がある。マニュアル作成者は、このことをよく分かっていなくてはならない。多くのユーザーは、包括的な機能や特徴が何十個あっても、そのようなことには無関心である。たとえば、開発者にとっては、もっとも大切な研究課題の1つに“パラメータ（媒介変数）をどのように定義付けるか”ということがあるが、医者や倉庫管理者は、そのようなことには興味を示さないだろう。

ここにパラドクスが生じる。多機能ソフトには、たとえば大型汎用コンピュータで動くデータベース管理ソフトや、一般的なパソコン用に設計された競合する各種“スプレッドシート”ソフトなどがある。しかしこのような多機能ソフトの大半が使用されるのは、**適応実務**においてである。ソフトを開発した個人あるいは企業は、その多機能性を手放しで誇りに思っているし、コンピュータに熟達している洗練されたユーザーは、ソフトの利用法をいくらでも考え出すことができるだろう。ところが、大部分のユーザーは、「自分たち」のプロジェクトの進め方、「自分たち」の問題の解決方法、「自分たち」の業務の改善策を知りたいと思っているのである。

●ユーザーには固有の利益がある

マニュアルに盛り込まなくてはならないのは、次の3種類の情報である。

- 「ユーザー」の職場にシステムを導入する際に必要となる情報。
- 「ユーザーが自分で」アプリケーション・プログラムを作成するためにシステムをどのように利用したらよいかを、「ユーザーの実際のニーズ」や利益をもとにした例を用いて指導した情報。
- または、「特定のユーザー」の問題解決のためにあらかじめ作られた「アプリケーション・プログラム」。

この3種類の情報のうち、今日の開発者にとって、最初の2つは3番目に比べてはるかに有用であり、3番目には何ら発展性がないように感じられるかもしれない。驚くべき多機能性を誇るソフトを、まるで1つの機能しか果たさないアプリケーションのように扱ったマニュアルを書きたがる理由がどこにあろうか。そんなマニュアルは、ソフトの価値を下げはしないだろうか。

否、まったく逆である。ユーザーにソフトの使い道をよりはっきりと示すことになり、した

がってマニュアルは、より使いやすくなるのだ。読者には、独自の使用目的や利益を持つ権利が与えられるべきである。多機能のソフトやハードを、スピーカーやテレビゲームと同じように、1つの機能を果たす道具にすぎないものとして扱っても構わないはずである。

●並列型と直列型

図表3.4のアウトラインの比較を見て欲しい。2つのタイプの間には、共通の項目がたくさんあることにお気付きだろうか。双方のスタイルにはいくつか興味深い違いがある（このことは後で述べる）が、主な違いは、Aは並列型であり、Bは直列型であるということだ。Aを用いるユーザーは、ひっきりなしにスキップ（読飛ばし）やループ（循環）をしなくてはならず、このようにマニュアルが使いにくくなれば、ソフトの有用性も引き下げられることになる。Bの方は、実務に即して構成されている。使ってみると、Bの方がずっと信頼できるはずである。

図表 3.4 ＜並列型マニュアルと直列型マニュアルの対比＞

Aタイプ（並列型）	Bタイプ（直列型）
1.　システム管理	1.　システムを導入する
1.1　機密保護機能の省略時解釈	1.1　添付ディスクのバックアップを取る
1.2　装置構成の定義	1.2　会社の機密保護に関する規則を定義する
1.3　ファイルの初期設定	2.　ファイルを作成する
2.　ファイル管理	2.1　システムに科目体系を登録する
2.1　ファイルの定義	2.2　仕訳伝票を入力する
2.2　ファイルの読込み	2.3　一連の"ファクター・プランニング"プログラムを書く
2.3　ファイルのリンク	3.　適応業務
2.4　ファイルの更新/保守	3.1　損益分岐点により損益を分析する方法
3.　入力準備	3.2　年度別原価差益を比較する方法
3.1　ワークシート	3.3　収支を予測する方法
3.2　データ入力	3.4　予算案をシミュレートする方法
3.3　データ編集	3.5　収益実績をシミュレートする方法
4.　出　力	4.　報告書作成
4.1　印　刷	4.1　傾向チャートを作成する方法
4.2　図表の印刷/点描	4.2　分担チャートを作成する方法
4.3　保　存	4.3　比較チャートを作成する方法
付録I　代替システム構成	4.4　各種の表を作成する方法
付録II　出力サンプル	
付録III　エラーメッセージ一覧	

訳注＊ Run-Book：特に事務計算などでは、使用するデータ、入出力装置、プログラムなどが、処理によってすべて異なる場合がある。こうした計算処理を，いつでも誰でも正しく行えるよう、その手順や扱い方が書かれた指示書。

3.5　激論!：有用性か経済性か

マニュアルを使いやすくしようとすると、繰返しや重複、浪費（出版部門の責任者はそう呼ぶだろう）につながる数多くのことをあえて行なわなければならないことが分かってくる。有用性と経済性の目指すものはしばしば対立する。その対立は基本方針の設定と充分な話し合いを経て、解消されるべきである。

●スペースを節約するな

会社でマニュアル作成を指揮するのは、出版部門の管理職が多い。皮肉なことに、管理職がまず優先するのは、たいてい「生産の経済効率」であり、マニュアルなどにかける費用は、ぎりぎりに抑えなくてはならないものである。

しかし、その結果マニュアルの出来があまりよくなかったために大きな無駄が生じて、一時的に節約したコストなど問題にならなくしてしまう場合もある。また、マニュアルの作成や発行や保管にかかる費用を削減すると、マニュアルの読みやすさまで削減されてしまう。たとえば、マイクロフィルムや細かい印刷文字を喜んで読むユーザーがいるだろうか。「出版社は、同じことを二度印刷せずにすんでさぞかしよかったろう」といいながら、本の2つの章を喜んで行ったり来たりする読者などいるだろうか。

情報の入る「スペース」（本の場合はページ数）を減らすのによく用いられる方法を考えてみよう。小さい文字で印刷したり、上下左右の余白をなるべく狭くすることは、ページ数を減らす一番簡単な方法であり、印刷、発送、保管にかかるコストも削減されるだろう。たとえば、ある米国大手の技術情報サービス*は、シングル（1文字分の）スペースと、3/4インチのマージンを力説している。結果は、ぎっしり詰まったほとんど読めない報告書となる。

同様に編集者や発行者の多くは、余白部分や半分空白のページを嫌う。しかし、本書では、各セクションの冒頭を必ず新しいページに持ってくる、というやり方を提唱している（またそれを実践している）。結果は、ゆったりした読みやすい文書となる。

●redundancyは信頼性を意味する

もっとも論議の的になるのは「redundancy」である。これは学生時代に書いたレポートや論文に対する批判としていわれた覚えがあるだろう。しかし、このredundancyは、必ずしも批判のためだけの用語ではない。エンジニアリングにおけるredundancyは、慎重を期すための“バックアップ”手順や手段のことを指すのであり、基本のデバイスが壊れたり、うまく機能しない場合でも、システムを動かしておけるようにする技術である。スーツに縫い込まれた替えボタンから、原子力発電所の冷却ポンプを作動させるのに使われる3機ないし4機の予備電源に至るまで、発想は同じである。redundancyは信頼性を意味する。

通信工学では、redundancyとは、通信チャンネルにおけるノイズやエントロピー*を補正するものである。雑音の多いチャンネルから正確な信号をキャッチする一番間違いのない方法は、何度も発信することである。パイロットが管制塔と話すときに「はい了解」と繰り返すのも、電信による資金為替が最低2回に渡って送信され、パリティ・チェック（奇偶検査）*が行なわれるのもこのためである。

場合によっては、文字が大きく、上下左右の余白が広く、章の終わりに空白が取ってあることさえ、情報が雑然として筋道が分かりにくくなることが回避できるという意味で、印刷技術から見たredundancyの発現形態であるといえる。redundancyがもっと明確な形で現われるのは、実際に「繰り返される」（同じ文や図表が明らかに2ヶ所以上で用いられる）場合である。

しかし、出版部門の管理職にとっては、こんなことは考慮の対象外のことだろう。さらに興味深いのは、技術雑誌の編集者のほとんどすべてが、すでに本文で述べたことを絵や図に描き換えるというやり方をいやがるということである。文章でうまく表現できたことを繰り返し図に表わす必要があるのか、と彼らはいうだろう。

●読者によって書き分ける

マニュアルの短期的なコストが気になって仕方のない人々は、図表やイラストは、必要でない限り、つまり文章だけでは表現できないものを示すとき以外は用いるべきではない、と本気で信じているようだ。しかしながら、よく出来たマニュアルでは、絵や図で文が“バックアップ”されるのだといいたい（その逆もいえる）。というのは、言葉を得意とする読み手と図表を得意とする読み手がいるため、もし読者のニーズに応えて、マニュアルが正しく読まれるようにしたいのなら、双方の読み手のために書くべきだからである。

故意の繰返しやredundancyがすべて望ましく効果的である、というのではない。それがあまり適切でない場合もある。くどすぎることもあるし、使いやすさという基準に合わなくなることさえある。また、redundancyの度合いと形式は、マニュアルとユーザーの関わり方にもよる。大学院の学生向けのテキストに比べると、オペレーション・ガイドは、より多くのredundancyが必要だ。そして、経験のない事務員や作業員向けのオペレーション・ガイドには、経験者向けよりも多くのredundancyが必要である。

●目先の節約にとらわれるな

まれに、運よく経済性と有用性を兼ね備えている場合がある。しかし、マニュアルを使いやすくするという条件のほとんど（たとえば、丈夫で厚い紙、カラー印刷など）は、とりあえず高くついて無駄のように思われる。しかし、それは目先だけのコストを見た場合である。長い目で見ると、効率性や生産性が高まって、何倍ものコストが節約できる。大局的な見方をすれば、マニュアルを読みやすくするための出費は、フィールド・サービス、ホットライン、ユーザー・トレーニング、トラブル対策など、さまざまなユーザー・サービスにかかるコストを省く、すなわちすでにそれらを含んでいるといえる。

優れたマニュアルは何倍も元が取れるはずだ。マニュアルの質を高める功労者は、四半期の予算にではなく、「総体的」なコストに気を配る人であり、結局そのような人を抱えているところが、最良のマニュアルを作る会社なのである。

原注＊技術情報サービス：米国には、新しい記事や技術報告書などに関する目録、要約、抜粋などを提供する組織が数多くある。利用者の会費によって運営される民間企業の場合もあり、安い費用で誰でも利用できる公共機関の場合もある。

訳注＊エントロピー：その通信チャンネルで送ることのできる情報量の尺度の１つ。redundancy、ノイズ、エントロピーの３つの尺度で、その通信路の情報量を測ることができる。

＊パリティ・チェック：データの信頼度を上げるために余分な情報を付加する方法。

3.6 最終テスト：信頼性と保守性

有用性のチェック項目（マニュアルがスキップ、ブランチ、ループ、回り道などをどの程度克服しているか）は、恣意的なものでも感覚的なものでもない。それは、システムを作動させたり、サポートする場合の費用に直接関わるものである。使いやすいマニュアルとは、「信頼性」と「保守性（メンテナンスのしやすさ）」に優れたものでなければならない。つまりうまく使えないことがなるべく少なくなるように書かれたものであり、また、問題がある場合の修正や調整が簡単にできるものでなければならない。

●マニュアルの信頼性とは何か

文書作成を1つのシステムと考え、各マニュアルがシステムの構成要素、あるいはデバイス（装置）であるとするならば、各マニュアルはできるだけ信頼性の高い、メンテナンスのしやすい形で作成すべきであろう。

ところで“マニュアルの信頼性”とは何であろうか。マニュアルが“失敗”するとは、どういうことなのだろうか。平均故障発生間隔（MTBF：工学におけるもっとも一般的な信頼性測定法）*によって、マニュアルが評価できるのだろうか。マニュアルが“故障”することが本当にあるのだろうか。

マニュアルが原因でユーザーやオペレーターの仕事ができなくなった、つまり**誤操作、故障、中断などの直接的な原因がマニュアルにある**場合は、マニュアルの失敗といえる。したがって、マニュアルの失敗は、書かれなかった情報、間違った情報、曖昧なあるいは矛盾した情報によって引き起こされる。また、必要な情報を探すのにかなりの努力が必要になるような形で内容の配列がなされている場合も、スタート地点を間違えたり、無駄な努力を払ったり、そのマニュアルでは処理できない問題を“その場しのぎ”の方法で解決しようとしたりするという障害が引き起こされる。

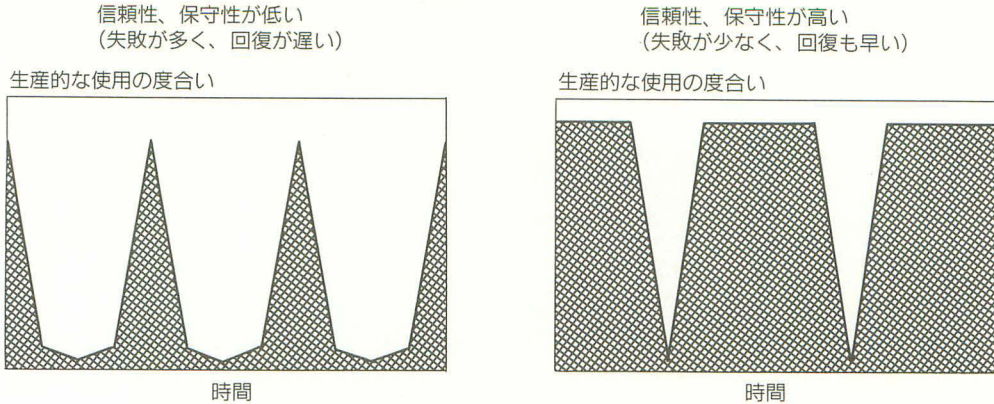
誤操作と作業中断という2つの失敗のうち、どちらの方が高くつくのかを判断するのは難しい。作業中断の方が一般的ではある。なぜなら職場では“作業中断時はコーヒータイム”となってしまうため、金を支払って従業員を休ませることになってしまう。しかし、おそらくもっとも高くつく失敗は“読者に誤解されても仕方のない”ようなマニュアルの記述によって、ファイルの破損やその他の深刻な破壊が発生することである。

●有用性と信頼性の関係

有用性を測る指標が正当なものであるということは、次のようなことからもよく分かる。1冊のマニュアルの中では、スキップ（読飛ばし）やループ（循環）による複雑さと、エラーや故障の起こりやすさとの間には、はっきりした因果関係がある。マニュアルに筋道が多ければ多いほど、間違った道を通る可能性も高い。また、マニュアルを読む際に不連続の動きや選択肢が多ければ多いほど、間違った動きや選択をする可能性が高い。

この考え方は、あまり経験のない読者、特にマニュアルを読む能力があまりない読者には、間違いなくあてはまる。しかしこの原則が、難しいマニュアルをうまく読めない読者にだけあてはまると勘違いしてはならない。入り組んでいて、信頼性に欠けるマニュアルを読み慣れている読者もいるが、決して好きで読んでいるわけではない。

出来の悪い雑然としたマニュアルが、筋道の一貫したGOTOのないマニュアルと同じように安心して読めないことは明らかである。

信頼性は1つの目標とみなし得る。読者X（マニュアルの内容を読み取るのに苦労している）

図表 3.6　＜信頼性と保守性の向上＞

を扱おうとしているマニュアル作成者は、目標を高く置いた方がよい。読者Y（複雑なマニュアルに慣れていて怖がらない）を対象とするライターは、目標を比較的低く置いてもよいだろう。

●信頼性の高いマニュアルはメンテナンスもしやすい

信頼性は、その目標をどのレベルに置こうと、マニュアルが“失敗”する頻度をもっともよく示すものだ。この考え方は、「メンテナンスのしやすさ」につながる。つまり、ユーザーがミスを修正して、再び作業にとりかかれるまでに、どのくらいの時間を要するか、ということである。錯綜やループ（循環）はユーザーの仕事を複雑にするが、同様にマニュアルの不備な点を見つけたり、変更したりする作業をも複雑にする。

マニュアルが入り組んでいて使いにくいと、潜在的で解決が困難な問題が起こる。たとえば、**あるページで1ヶ所変更すると、別の所で信じられないような、まったく予期できないエラーが発生する**ことがある。マニュアルからある図表を削除した場合に、その図表に関するあらゆるコメントを間違いなく削除できるだろうか。ある報告書の名前を変更した場合に、それが現われる個所をすべて間違いなく変更できるだろうか。制限事項を緩和した場合に、その制限に関わる操作すべてに、間違いなく言及できるものだろうか。

マニュアル自体を変更することも、やはり混乱を引き起こす原因となり得る。マニュアルの中には、すんなりと補足や修正を受け入れるもの（メンテナンスしやすいもの）もあるが、それを拒むものもある。標準マニュアルをたびたび改訂しているDPセンターの中には、どのマニュアルを取っても、同じバージョンのものは2つと見あたらないというところもある。

訳注＊平均故障発生間隔：Mean Time Between Failure。故障が発生してから、次の故障が起きるまでの時間の平均。

第II部 マニュアルへの構造的アプローチ

第4章

“土壌”：マニュアルはいかに書かれるか

4.1　マニュアルを“書く”2つの方法

マニュアルを書くには、大きく分けて2つの方法がある。1つは、「執筆・構成する」という方法（“作家”が原稿に向かってそうするように、文章や段落をいろいろ工夫してみるやり方）である。もう1つは、「企画・設計する」という方法（段階を踏んで徐々に詳しくなっていき、ついにはマニュアルが“必要でなくなってしまう”ような一連の仕様書を作成するやり方）である。

●プロはプランニングを行なわない？

“ライター”という言葉から人々が思い描くイメージは、原稿用紙の上の1行1行の文章に対して一所懸命取り組んでいる人ということだろう。そこには、ひらめいてなぐり書きする、そして猛烈に推敲し、手直しする、また手が止まって次のひらめきをじっと待つ、といった一連の作業がある。

コンピュータのプログラマーにも、同じような一般的イメージがある。つまり、試行錯誤、着想からの推論、天才的なひらめきなどを通してキーボードをたたきながら考える人、といったイメージである。

このような紋切型のイメージのプログラマーやライターは、実際に存在している。彼らにとってのプログラミングおよびマニュアル作成とは、ルーズで、非計画的で、“芸術的な”やり方で片付けられるような、かなり単純なプロジェクトである。実際、プログラマーやテクニカルライターは、何のためらいもなく、ほとんど明確化されていないプロジェクトに、のべ3、4ヶ月も費やして、よい結果が出ることを期待するという意味では極めて特異な存在である。

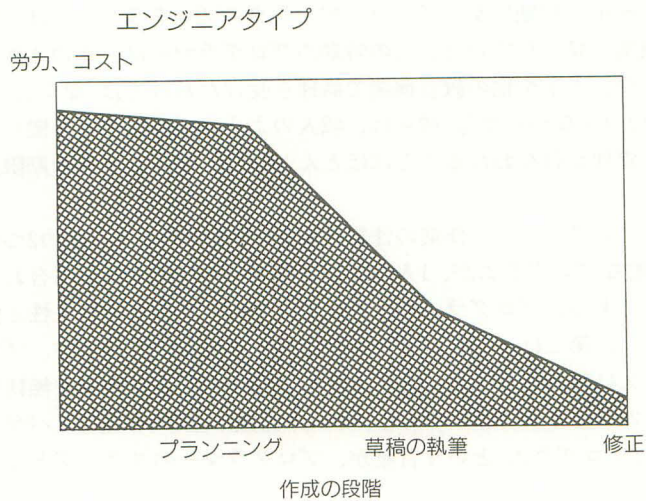
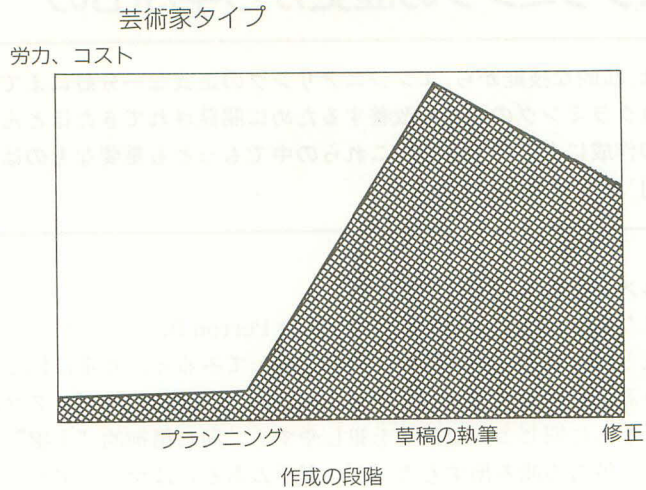
もちろん、どの分野においても、まったくプランニングを行なわずに仕事をするプロが、ごくわずかだが存在する。このようなプロは、上述のプログラマーやライターと同様、仕事を過大視したり、逆に単純化したりする。マニュアル執筆者（プロのテクニカルライターも含む）のほとんどは、いまだに他の分野のプロに比べ、仕様書なしで仕事をするケースが多い。分厚い本を書くのに、ありふれたアウトライン（小学校で習った程度の方法で書かれたアウトライン）を作成するだけでは、エンジニアリングの利点を活用したことにはならない。

●芸術家タイプとエンジニアタイプ

図表4.1に見る通り、いわゆる芸術家タイプのマニュアル作成者は、プランニングという作業に、ほとんど重きを置かない。彼らは、まず草稿の執筆に力点を置き、その後つじつまの合わない部分に“ツギをあてること（パッチング）”に全精力をつぎ込む。

また、図表4.1はいわゆるエンジニアタイプのマニュアル作成者が、どのように力を配分するかを示している。彼らの努力は、ほとんどプランニングに注がれる。つまり、どんなマニュアルが必要かの定義、どんなマニュアルにすべきかという設計、マニュアルのモデル構築、といったことである。彼らにとって原稿を書くということは、企画・設計を単に実現したものにすぎず、製品の創造そのものではない。

これら2つのタイプの間に、上記以外の違いが現われるのは、他の人がマニュアル作成に関与したときである。芸術家タイプは、作業が“一段落する”までは、草稿を他人に見せようとはしない。反対に、エンジニアタイプの手になるマニュアルは、さまざまな作成段階において、幅広く討議され、評価され、見直される。このため、出来上がったマニュアルには、執筆者のエゴが必要以上に大きく反映されることはない。

“芸術家”という名前から、芸術家は例外なくプランニングを行なわないと想像しないで欲

図表 4.1　<マニュアル作成における力配分>

しい。また“エンジニア”という名前から、エンジニアは例外なく系統的に仕事をすると思わないで欲しい。現実には、多くのプロのライターが、草稿を書き始める前にきちんとした計画を立て、多くのエンジニアが、無計画な試行錯誤の中で問題を解決しているのである。もっとも、この試行錯誤を“プロトタイピング（試作）”と呼んで、作業を系統化しようとする傾向がコピュータ・プログラマーにはある。芸術家とエンジニアを区別しようという提案は、むしろ以下のことを強調することを目的としている。つまり、ある種の業務は、芸術家でもエンジニアでも、どちらの“土壌”の人間でも行なうことができるが、プロジェクトが複雑になり、金銭的リスクが大きくなった場合、このような作業は、芸術家よりもエンジニアに任せられるべきであるということである。

4.2　プログラミングの歴史から学ぶもの

プログラミングは、私的な技能から、エンジニアリングの正式な一分野にまで発展してきた。幸運なことに、プログラミングの技法を改善するために開発されてきたほとんどすべてのツールは、マニュアルの作成にも応用できる。これらの中でもっとも重要なものは、「トップダウン（下方展開型）作業」と「テスト」である。

●スパゲッティからメンテナンスのしやすさへ

1979年に発行されたComputerworld誌で、Robert Perron氏は“コンピュータが出現したころのプログラム作成と、今日のマニュアル作成を比較してみると、非常に似ている点がいくつかある”と述べている。1980年代のマニュアル作成者は、1960年代のプログラム作成者とよく似ている。彼らは、ともに同じような誤りを犯しやすく、同じ精神的“土壌”を持っているようである。このため、彼らの産み出すもの（プログラムあるいはマニュアル）は同じ欠点を持つのである。

1950年代末期から60年代初期にかけて、コンピュータのプログラミングは、特異な仕事、あるいは風変わりな職業と見られていた。この時期のプログラマーは、その才能と情熱によって特徴付けられていたが、大学や他の教育機関で訓練を受けたわけでは「なく」、一般のホワイトカラーのようには扱われ「なかった」。彼らは、職人のように一匹狼として働いていた。彼らの仕事に対して厳しい管理が行なわれることはほとんどなく、彼らは予算や期限にもあまり悩まされることはなかった。

このような過去のプログラマーの作業の性質を変えたのは、何よりも次の2つのファクターである。第一は、一般的なプログラムが、1人のプログラマーの作業では間に合わないほど大規模なものになったことである。プログラムの巨大化が、プログラマーの自主性と優雅な孤独に終止符を打ったのである。第二は、プログラミングに関する費用の大部分が、プログラムの作成からそのメンテナンスに移行したことである。そしてプログラムの大半が無秩序で、入り組んでいて、メンテナンスのしにくいものである、という認識が生まれた。“スパゲッティ・コード”（もつれて入り組んだプログラム）という言葉が、プログラマーのスラングとなったのもこのころである。

一方、今日の組織では、マニュアルの「ライター」は、他のホワイトカラーと違った待遇を受けている。マニュアルライターは、ほとんど管理されず、予算の問題で頭を悩ますこともほとんどない。たとえば、**今日の多くの企業は、マニュアル1ページの執筆に対するコストを見積る術を知らない。**

しかし、まさに複雑さ、規模、メンテナンスといった問題によって、昔のプログラム作成手法が時代遅れの方法となったように、同じような問題がこれまでのマニュアル作成手法を時代遅れにしつつある。商品が市場に出るのと同時にマニュアルが出版されるのであるなら、マニュアルの執筆は、スケジュール化され、予算の制約を受け、管理されなければならず、また並行して作業にあたれるライターのグループが必要である。そして、2、3ヶ月ごとのシステムの改訂にマニュアルも歩調を合わせるなら、変更の容易なマニュアルでなければならない。

●プログラマーの経験から学ぶもの

では、マニュアル作成者は、プログラマーが学んだことから何を学び取るべきなのか。まず第一に、ソフトウェア・エンジニアリングでもっとも大切な原則は、**開発サイクルの中で問題の抽出が遅くなればなるほど、問題の究明と解決に要する期間と費用が、指数関数的に上昇す**

るということである。最初のプランニング段階では2、3分、あるいは2、3ドルですむ問題解決が、企画・設計段階では数百倍、執筆・編集段階では数千倍、出版・運用の段階では数万倍に膨れ上がる。

最初は難しいかもしれないが、プログラマーであれ、マニュアル作成者であれ、第一線に立つためには、**進んで誤りを見つけ出す**術を身に付ける必要がある。テストを行なわなければ、有用で信頼できる技術は得られない。テストの役割とは、失敗を犯させることにある。テストで欠陥が見つからなければよいと思う人間、仕様書に対して討議がまったく行なわれなければよいと思う人間、アウトラインに問題が1つも見つからなければよいと思う人間、これらの人々は、誤りの発見が遅れる（早まるのではなく）のを期待する人間で、問題の解決に結局高い代償を支払わせるタイプであり、品質の低下を招く連中である。

したがって、マニュアル作成者がプログラミングの歴史から学ぶべきことは、以下に掲げる事柄を認識して、「トップダウン（下方展開型）作業」と「テスト」を実践することである。

1．誤りや問題の発見が早ければ早いほど、これらの修正は容易で、費用も安くすむ。マニュアル作成の初期段階での作業を個人化したり、非公式化したりすると、その代価は非常に高くつくことになる。
2．入り組んだ製品のもっとも重大な問題は、これを構成する各単位やモジュールの中にあるのではない。それらの連結および接合にある。したがって、マニュアル作成におけるコスト効果を高めるには、草稿執筆前に、マニュアルを系統立てて構築しなければならない。またマニュアルの各部を正しく連結する、マニュアルの各部において正しい筋道を立てる、内容に間違いがないことを確認するといったことが必要である。
3．マニュアル作成プロジェクトが系統立てられて企画・設計されていなければ、せっかく複数で作業を行なっても、1人で作業をするより時間がかかってしまう場合がある。したがって、マニュアルを急いで作成するときほど、詳しく書かれ、系統立てられたモデルに沿って作業を行なう必要がある。
4．複雑に入り組んだ製品やシステムを、開発の初期の段階においてぞんざいに設計してしまうと、メンテナンスとサポートを行なう際に、たいてい高くつくことになる。したがって、質が高く、メンテナンスのしやすいマニュアルを開発しようとしているときに、時間が足りないとか、費用が足りないとか文句を付けることは、ほとんどの場合、不当なことである。

4.3　マニュアル作成を効果的に進める5つのプロセス

マニュアルを作成する上で必要なプロセスとは、次のようなものである。まず、本当に役に立つマニュアル、真に求められているマニュアルとはどういうものかをつきつめていくこと。次に、ジャンプ（飛越し）やスキップ（読み飛ばし）や回り道などが、なるべくないようにすること。それから、明解さ、読みやすさ、信頼性などを向上させること。最後に、マニュアルをメンテナンス（保守）のしやすい形にすることである。

マニュアルに関して研究室での厳格なテスト（人間の諸要素に対する心理学的分析と正式なテスト規約にのっとって行なわれる）を取り入れ始めている企業は、アウトラインの中の欠陥よりも、完成した草稿の中の欠陥の方が扱いにくいことに間もなく気付くだろう。また、プログラムのモジュールに対して単体テストを行ないながらも、テスト後のモジュールの組立てと統合がうまくできないという会社の場合も、同じことがいえる。これらの会社が見逃していることは、書かれた後にではなく、書かれる「前」に、プログラムやマニュアルのモジュールが組み立てられ統合されなければならない、ということである。すでに述べた通り、マニュアルの執筆者と読者をもっとも悩ます問題は、モジュールやページに書かれた内容にあるのではなく、モジュール相互の連結および接合関係にある。

●5つのマニュアル作成プロセス

マニュアル作成のプロセスが、過去10年間のソフトウェア・エンジニアリングの成果から学ぶべきことは、以下の5点である。

1. **ユーザーのニーズと使いやすさの双方を“満たす”マニュアルを企画する。**書かれるべき内容－マニュアルや他の情報製品の－をユーザーのニーズに合致させるという方法が必要である。言い換えるならば、巨大な分量の1冊の汎用型マニュアルを作成するのではなく、特定のユーザーやオペレーターの特徴と、彼らのシステムに対する使用目的に合わせたマニュアルを作成することが必要なのである。もっと簡単にいうならば、マニュアルと他の情報製品を、どのように論理的に組み合わせるのかについて定義する。つまり作成される文書セット全体の論理構成を考えるのである。また、こうして定義された文書セットの各構成要素に書かれるべき内容は、「試案としてテストできる」ものでなければならない。そうすれば、計画がマニュアルやディスクに形を変える前に、作成計画に関する方針上の失敗が発見でき、プランを練り直すことができる。
2. **マニュアルや他の情報製品の中の、スキップ、ジャンプ、回り道を減らす。**近ごろの文書処理方式は、確かにテキストのさまざまなブロックを変更しやすくしてはいるが、一度出来上がってしまった草稿に対して構造上の修正を加えることが困難だという事実はやはり否めない。したがって、修正が難しくなる草稿完成の「前」に、マニュアル作成における構造上の欠陥を明らかにする効果的な手法が何か必要である。効果的なマニュアル作成プロセスにおいては、次第に記述が詳しくなっていくマニュアルのモデルが作られる。こうしたモデルが作られるのは、アウトラインと草稿の間（いわゆる芸術家タイプのライターがたいてい1人で行なう作業期間中）であるが、このモデルについては、「有用性」という明確な基準でテストを行なうことができる。
3. **ライターがチームを組んで並行して作業を行なう。**不備なマニュアルに対する言い訳としてもっともポピュラーなのは、マニュアル作成のために充分な準備をしようとすると、システムの完成やその販売が数週間から数ヶ月遅れてしまう、という主張である。しかし、

この主張はどう考えても、出来の悪いマニュアルが作成されたことの正しい説明にはなり得ない。むしろ、マニュアル作成には専門的手法が必要である、ということを示唆している。この手法とは、マニュアル執筆にチーム作業を導入し、明確に細分化されたマニュアルの各部分（切身）について、「並行して」作業を行なうことである。

しかし、用いる方法が正しくないと、ライター同士の共同作業も、マニュアル作成のスピードを遅くしてしまう。間違った方法を用いると、1人で書くより2人で書く方が2〜3倍も時間がかかってしまう。したがって、マニュアル作成を効果的に進めるには、執筆作業を「独立して作業できる」単位に細分化し、これらの単位を有機的に統合する必要がある。このように、作業を細分化し統合すれば、コストとスケジュールをあらかじめ定めて管理することができる。

もっと詳しくいうと、執筆作業の大部分を、作業量とコストの見積りが簡単にできる小部分に“分解”するのである。また、細分化された作業相互の連結および接合関係をモデルごとにあらかじめ定め、明確化し、テストすることが必要である。そうすれば、細分化された各部分が独立して執筆されることになり、執筆者同士が相談する必要もない。各執筆者は、マニュアルの全体モデルさえ頭に入れておけばよいのである。

4．**マニュアルや他の情報製品の、明確性、可読性(読みやすさ)、信頼性を高める。**草稿上の欠陥が重大であることはもちろんである。マニュアル作成の目標は、草稿の出来上がる「前」に方針上の問題をすべて解決し、構造上の欠陥をすべて修正することにある。したがって、効率のよいマニュアル作成においては、草稿が完成した時点では草稿上の問題しか残っていないはずである。ここで必要なのは、システムに関する不的確な表現の訂正、技術面でのちょっとした変更、意味の分かりにくい文節、曖昧模糊とした文章、混乱を引き起こすような分かりにくいイラストなどの修正である。

また、効果的なマニュアル作成プロセスには、編集に対する正式な基準がある。1人の人間の芸術的、直感的、あるいは“文体上の”嗜好には左右されない。たとえば“F(4)を続けて押してください”というようなプロンプト*は、曖昧で信頼性が低い。適切なマニュアル作成プロセスでは、編集者が主観的に妥当と思うか否かや、テスト段階で読者がそのプロンプトにとまどわされたか否かに関係なく、このようなセンテンスは洗い出され、修正される。

5．**メンテナンスしやすく、変更可能なマニュアルと情報製品を作る。**出来のよいマニュアルは、出来の悪いマニュアルよりも修正や追加補正が少なくてすむ。ただし、修正が必要なときは、修正プロセスが作成プロセスよりも簡単であることが必要で、混乱や誤った情報伝達を引き起こしてはならない。

訳注＊プロンプト：prompt（“促すもの”の意）。コンピュータが、画面に表示する指示。これが表示されている間は、コマンドなどの入力待ち状態になっている。

第5章

作成過程の構造化：マニュアル作成の5つのステップ

5.1 “構造化”とは何か

「構造化」という概念は、主に次の2つの点でマニュアル作成に適用できる。まず第一に、マニュアルの開発プロセスは“構造化プロセス”として特徴付けられる、という点である。第二に、マニュアル自体がしばしば“構造化文書”と呼ばれる、という点である。ただし今日、構造化という言葉が、あまりに気軽に、そしてあまりに不用意に使われているということは憂慮すべき事態であり、どこかで歯止めをかけて言葉を正確に定義しておく必要があるだろう。

「構造化」という言葉は、統一化、組織化といった言葉の同義語として日常会話の中でいい加減に使われることが多いが、むしろ私は、コンピュータ科学者やシステム・エンジニアが“構造化分析”“構造化設計”“構造化プログラミング”というような表現の中で使用しているのと同じ意味でこの言葉を使用したい。

これら3つの用語の中では、構造化という言葉は、次のような「構造化分析」に関する定義にみられるある種のプロセスないし手法と深い関係にある。

ある問題またはプロセスを公式にもとずきトップダウン（下方展開）方式で分解・分析して１つのモデルを作ること（このモデルは、その問題またはプロセスの完璧で正確な記述を提供してくれる）は…プログラミングのための基礎となる…

—Sippl and Sippl Computer Dictionary & Handbook (Indianapolis : Sams & Co.), 1980, p. 529

まず第一に構造化分析は優れて「形式論的」である。つまり、誰にでも理解でき、明快であり、普遍的なルールにもとづいたものである。さらにいえば、ここでいうプロセスが、もし勘に頼ったり、個人的なものであったり、ルールや指針を無視して進められるものであったとしたら、それは構造化されているとはいいがたい。

第二にそれは「トップダウン（下方展開）的」である。つまり、できるだけ広い視野のもとに、あらゆるインターフェイスを含んだシステム全体としてまずとらえられ、次にある一連のステップを踏んで、細部がそこに重ねられていくものである。そして、システム全体は細部の細かいレベルに至るまで、その妥当性が「テスト」される。

●解体する前に全体を見極めよ

情報産業に携わる多くの人々は、大きな概念をより小さな概念へと分解することがトップダウン分析の本質であると誤解している。「分解」という第三のキーワードが、この誤解を産み出しているようである。構造化分析には、分解作業（大きなものをより小さなものへと分化させること）が付きものであるが、それにはまずシステムの全体像がどんなものかを明確化しておく必要がある。構造化されたテクノロジーにおいては、各部分の内容が定義される前に、各部分はすでに、全体の中に組み込まれているのである。

●モデルは製品の改良を安上がりにする

上記の定義の中で、次にキーワードとなるのは、「モデル」である。簡単にいえば、構造化の方法では、まずある製品のモデルが製作され、次に製品そのものが製作される。なぜなら、完成された製品を改良するより、モデルを作って、それに修正を加えた方がはるかに安上がりだからである。

●ダイヤグラムとは何か

構造化分析はこのくらいにして、「構造化設計」にいこう。

(構造化設計は)システムのコンポーネントと、これらコンポーネントの間の内部相互関係を、もっとも実行可能な方法で設計する技法である。別の言い方をすると、(構造化設計は)明確に仕様表に書かれた問題を解決するために、どんなコンポーネントが、どんなふうに内部的に相互結合されればよいかを決定する過程である。

—Yourdon/Constantine 著「ソフトウェアの構造化設計法」(日本コンピュータ協会) p. 9

図表 5.1　＜マニュアル作成に構造化の方法をあてはめる＞

このような方法で設計された製品には、次のような2つのものしか含まれないことに注意して欲しい。すなわち、コンポーネント(構成要素)とそれらコンポーネント間の相互関係、モジュールとそれらモジュール間のインターフェイス、ノードとエッヂ、ユニットとそれらユニット間の連結などである。このように構造化された製品、あるいはシステムには、2種類の要素しかないため、たいていの場合、簡単なダイヤグラムで図式化することができる。このダイヤグラムは、モジュール（ノード、コンポーネント、ユニット）を意味する四角あるいは丸い枠と、そのつながり具合い（リンク、インターフェイス、エッヂ）を表わす矢印、または線によって構成される。

繰り返すが、なぜこのようなダイヤグラムを作成するかというと－特にマニュアルのプランニングの場合－**訂正するのに手間とコストがかからないうちに、欠陥や問題点をつきとめるため**である。

●構造化はコストやメンテナンスを軽減する

開発者にとってこの手法の本当のメリットは、究極的にはメンテナンスの段階にある。なぜなら、メンテナンスの段階では、モジュールやユニットの単位で置き換えや追加を行ないさえすればあらゆる変更が可能であり、また、設計を綿密に行なえば、これらの変更からどんな結果が得られるかが予測できるからである。

ここで強調したい点は、プログラムやシステムをコストのかからないメンテナンスのしやすいものにするために使われた構造化の方法が、マニュアルを考案し作成するという作業にもそのまま適用できるという点である。マニュアルが構造化されれば、プログラムやシステムと同様、コストやメンテナンスの面で利益が得られるはずである。「作成過程」を構造化しさえすれば、そうして作られた「製品」一つまりマニュアル一も必ず構造化されるはずである。構造化されたマニュアルは、小さなコンポーネントやモジュールの集まりによって構成されることになる。それらのコンポーネントやモジュールは、マニュアルができるだけ読みやすくなるような方法でつなぎ合わされることになるだろう。

5.2 “モジュール化”とは何か

構造化システムが、分節化されたモジュールの集まりであるように、構造化文書も分節化されたモジュールの集まりである。各モジュールの出来がいいと、全体的にバラバラな感じがなく、先の予想もつきやすい。このようにモジュール化されたマニュアルは、設計さえしっかりしていれば、各モジュールの組合わせが必要以上に複雑になることはない。

一番紋切り型の定義によれば、「モジュール」とは、小型で独立していて、ある1つの機能を果たす単位であり、自分より大きい単位の一部をなすものである。この定義は単純すぎてかえって分かりにくい。モジュール化という作業は、とらえどころのない、ほとんど神秘的とさえいえる性質のものである。設計者や技術者は、作業が煮詰まってくると、モジュール化とは何かを“感じる”ことはできるが、適切な言葉でそれを定義することは難しい。そこで、上記の定義の各部分を順番を逆にして考えてみよう。

●モジュールはある１つの機能を果たす。モジュールは自分より大きいものの単なる部分ではない。自分自身「1つの機能を果たす」。その機能が何であるかは明確に説明できる。1つのモジュールは正確にある1つの処理を実行する。その処理とは、あるデータをより有用性の高い形態に変換することである。さらに、よくできたモジュールは、たいてい「総体としての1つの」処理を実行し得る。たとえば、ある1つのファイル内の仕訳データを未払い、前払いなどに分けて並べ換える処理がそうである。また、モジュールの出来がよいと、処理の結果を予測できる。同じ条件下で行なわれた処理は、必ず同じ結果を産み出すはずであり、そのモジュールには、そうした入出力形式を変えるような“内部経路”はいっさい含まれない。

●モジュールはそれぞれ独立したものである。各モジュールは、他のモジュールとの前後関係にいっさい依存しない。ある特定の機能を果たすモジュールは、他の文脈に置かれても、まったく同じ機能を果たす。ある意味では、各モジュールは、モジュールの出入れができる大きなセット（集合体）、あるいはライブラリの一部をなすこともある。結局のところ設計者は、手に入るモジュールのリストから目ぼしいものを物色し、新たに組み合わせさえすれば、システムや製品を簡単に構築することができる。

●モジュールは小型である。上記の定義の一番曖昧な点は、モジュールのサイズに関する部分である。各モジュールがある1つの機能を果たすといってしまうと、モジュールのサイズを正確に限定することはできなくなる。結局、モジュール化されたソフトウェアを使用する場合、ある1つの機能なり概念なりが1つのモジュールに収まるか否かをいつまでも議論することは、時間の浪費であると覚悟せざるを得ない。そこで、構造化の方法論を用いる人々のほとんどは、実質的な目安として、**モジュールの最大サイズをはっきり決めている。**データ処理においては、モジュールのサイズは、たいていプログラムのステップ数によって限定される。文書作成においては、ページ数が目安となる。

システムやマニュアルをモジュール化する醍醐味の1つは、モジュールのサイズを操作することである。モジュールのサイズを大きくすると、各モジュールの凝集度は低く（1つのモジュールが複数の機能を持つように）なる。逆にモジュールのサイズを小さくすると、モジュール間の連結や接合関係が複雑になる。モジュール化されたマニュアルでは、モジュール間の連結は、同じ本の違うページとの関連性として現われてくる。したがって本書の中心テーマは、モジュールの凝集度を高める代わりに、モジュール間の連結を複雑化しないという設計方針が、使いやすいマニュアルの開発に直接役立つことを証明することである。

マニュアル作成を上記のように取り扱う完成された方法論は、現在 Hughes Aircraft Company で働く文書作成の専門家たちが開発した方法論に、その共通性を見出すことができる。彼らのやり方は、映画産業からヒントを得た“ストーリーボード作業”という形式である。彼らの作る“モジュール”は、見開き1ページ（連続した2ページ）にきっちり収まるようになっている。これについては、彼らの次のような草分け的な業績に詳しく述べられている。

Tracey, J. R., Rugh, D. E., and Starkey, W. S. *STOP: Sequential Thematic Organization of Publications*. Hughes Aircraft Corporation: Ground System Group, Fullerton, CA, January 1965

このテーマをジャーナリズムが最初に取り上げたのは、以下の中でである。

Tracey, J. R., "The Effect of Thematic Quantization on Expository Coherence." *IEEE International Convention Record*, Paper 9.4, 1966

マニュアルのモジュール化は、マニュアルの読者にとって有益であるのみならず、設計者やライターにとっても有益である。アウトラインをモジュール化（“構造化”）することから作業を始めることにより、設計者はマニュアルのサイズとコストを見積ることができる。さらに、マニュアルが正確かつ読みやすいものであるかどうかをテストする作業を同時に行なうこともできる。その上、マニュアル作成という長い複雑なプロセスを、それぞれ独立した小さい作業の集まりに分解して、大勢の人間に執筆の分担を割り振り、それぞれ個々に並行して作業が行なえるようにできる。

マニュアルのモジュール化は、また“執筆者”にとっても有益である。この場合の執筆者とは、マニュアルの“原料”となるものを作成するのに必要な人々である。モジュール化されたマニュアルの場合、これらの人々をそのまま“初稿の執筆者”と呼ぶことができる。なぜなら彼らは、それぞれ自分がどれだけ書き、どのようなポイントをカバーすべきかを正確に把握しているからである。

“芸術家”を自認して単独で仕事をするライターでさえ、モジュール化の方法から利益を得ることができる。彼らはモジュール1つ1つに対して個別に仕事をすることができる。なぜなら、そうして書かれた小さい切れ切れの原稿の断片が、最終的にはうまく1つにまとまることをよく知っているからである。しかし、モジュール化の方法論から特に利益を得るのは、マニュアル作成を管理する人々である。モジュールによってマニュアルの企画、執筆、編集、製作などを行なうと、責任者のコントロールの質を根本的に高めることになる。

ここでもっとも重要なことは、モジュール化の方法によって効果的に設計されたマニュアルは、ユーザーにとって考えられる限りもっとも読みやすく“親しみやすい”マニュアルであるということだ。事実、モジュール化されたマニュアルを読んだ読者は、執筆者や開発者の個人的な思い入れによるあらゆる論証を上回り、マニュアル作りのコンセプトそのものに好感を示すことが多い。モジュール化されたマニュアルを好まないテクニカルライターを私は何人か知っているが、それを好まない読者にはお目にかかったことがない。

5.3 マニュアル作成過程の全体像

マニュアル作成を5段階に分けると、次のようになる。1.分析(どんな情報製品が求められているかを明確に定める)　2.設計（マニュアルのモデルをいくつか構築し、それらをテストする）　3.執筆・編纂（初稿をすべて書き上げる）　4.編集（“言葉のバグ”と技術的な誤りを取り除く）　5.保守（変える必要のある部分を変更する）

システムを開発する上で必要なサイクルやプロセスがあるように、マニュアルを作成する上でも、それにふさわしいサイクルやプロセスがある。それらを以下に掲げる。

1．**分析**：ユーザーやオペレーターが、どんなマニュアルや他の情報製品を必要としているかを明確に定めること。システムの場合、この分析作業が、開発段階の早い時期に行なわれれば行なわれるほど、明らかにそのシステムの品質は向上する。マニュアルの分析は一プロジェクトが大規模になると、出版計画の段階でようやく取り沙汰されることが多いが一理想的には、システムそのものの開発プランの一部として取り上げられるべきである。しかし、マニュアルに対するニーズの分析を行なうのに遅すぎるということはない。とにかく、どんなマニュアルが求められているかを分析せずに書かれたマニュアルは、たいてい失敗する。

2．**設計**：マニュアルあるいは他の情報製品ごとに、詳細なアウトラインを作成すること。そのための第一段階として、まず従来型のアウトラインを作成する。しかし、これはあくまで“構造化アウトライン”と“ストーリーボード”を作成するための1ステップである。つまり、マニュアルの初稿を書く前に、テストやチェックができるようなマニュアルの作業用モデルを作成するわけである。マニュアル作成の構造化や組織化を行なう上での重要な問題は、マニュアルの初稿が書かれる前に解決されていなければならない。

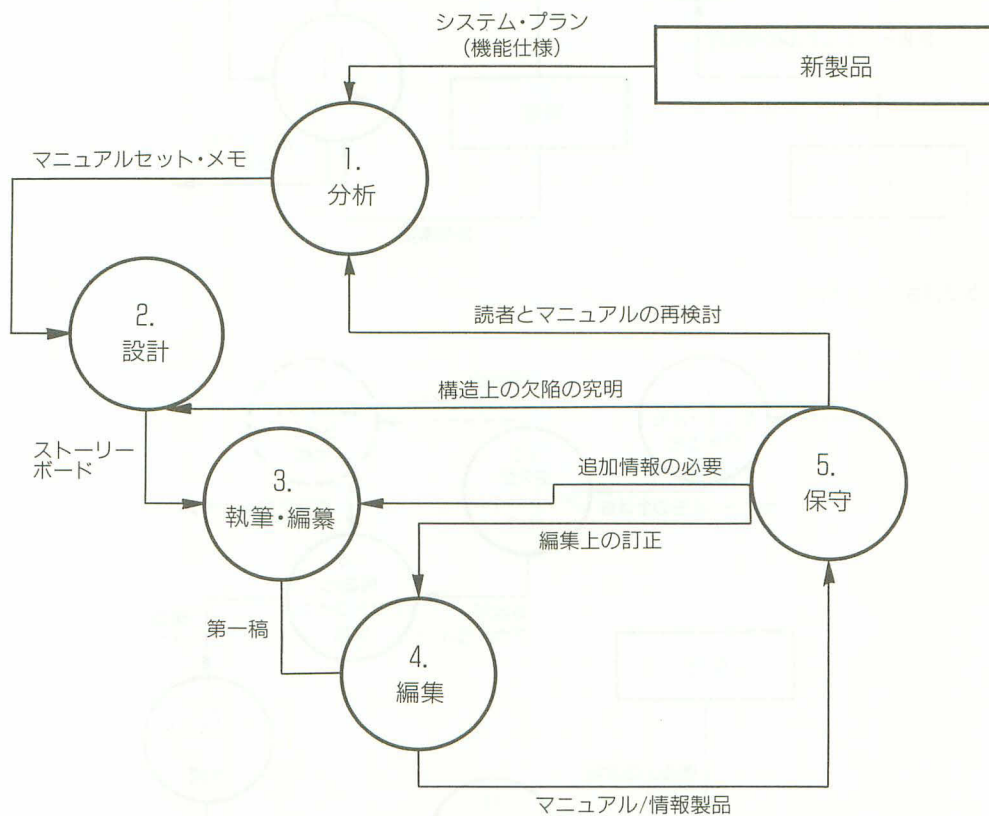
3．**執筆・編纂**：ストーリーボードを作業プランに変換し、初稿を書き上げること。マニュアル作成の構造化を目指す場合、初稿を書くことは構造化プログラミングにおけるコーディング作業に少し似ている。つまり執筆者には、厳密に立てられた作業プラン(“ストーリーボード”）に沿って、抜けている細部を埋めていく、という作業が最低限要求される。

4．**編集**：明快さ、正確さ、そして“読みやすさ”、つまり、テキストを読む上での容易さ、という観点から初稿をテストすること。このステップでは、言葉遣いや文体の問題は、審美的な事柄とは縁遠い、ということを忘れてはならない。このステップは、むしろマニュアルを使いやすいものにする、言い換えれば“失敗”させないための編集上の原則を適用することである。ここでいう“失敗”とは、オペレーターや読者が、マニュアルのバグによって、システムを操作できなくなることをいう。多くの場合、このステップの最終段階では、“本物の”読者に、実際にテストしてもらうことになる。

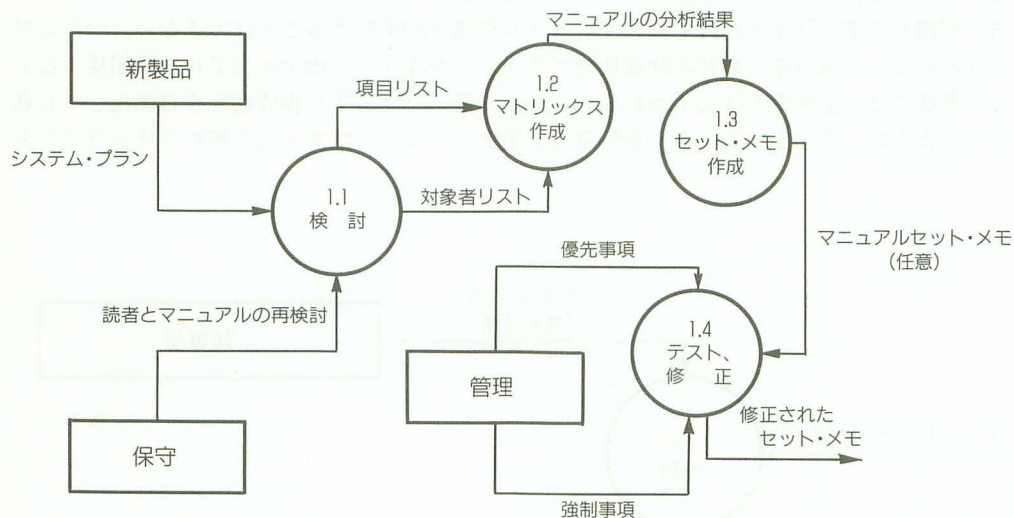
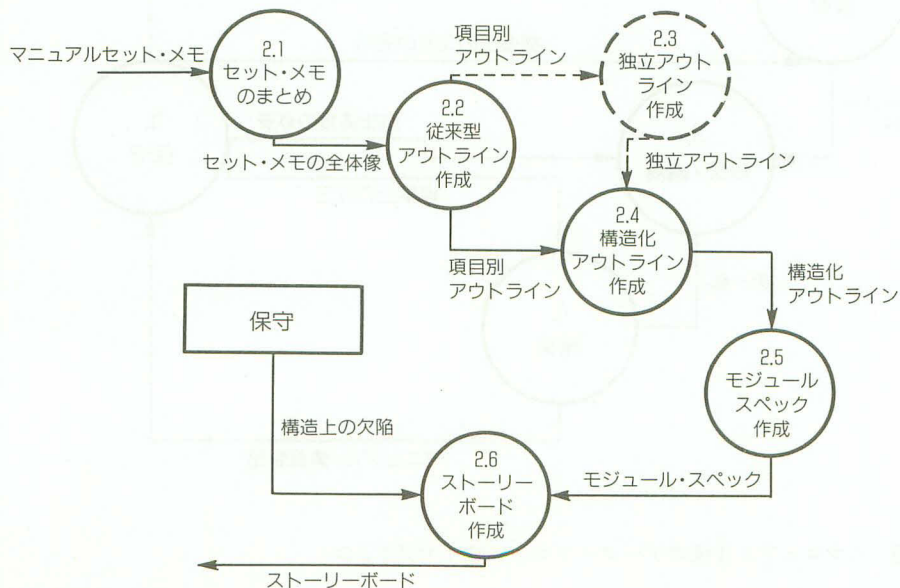
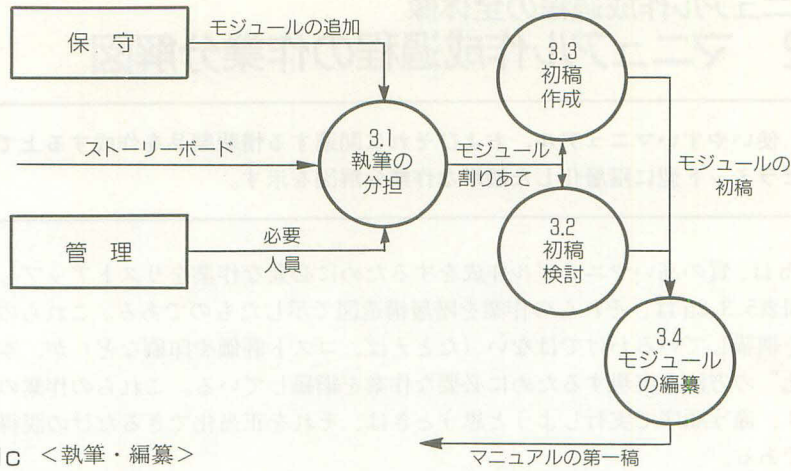
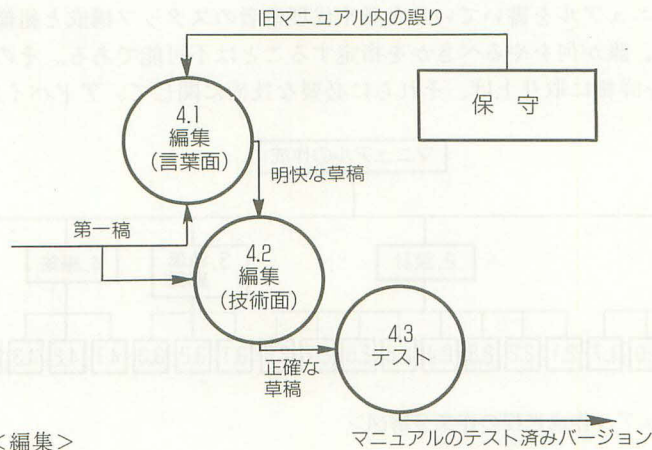
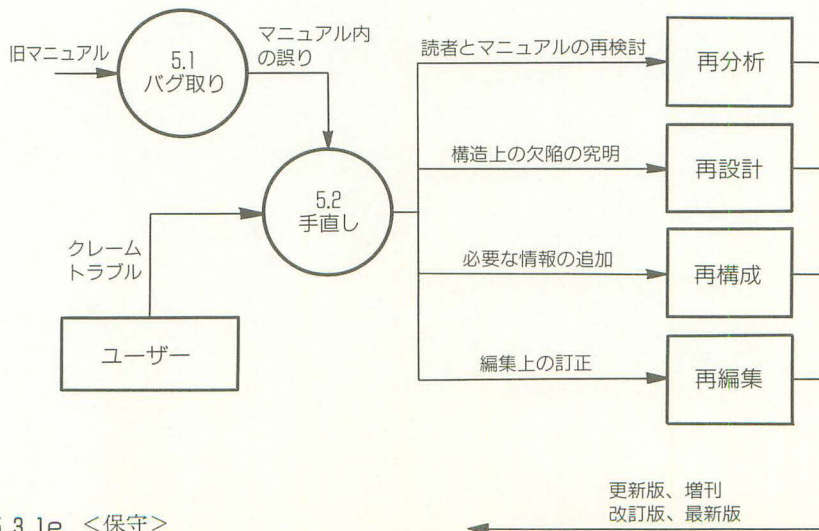
5．**保守(メンテナンス)**：情報製品の中のどの部分を変更する必要があるか、また変更をいつ実施するのが適当かについて考察すること。どんなマニュアルでも（例外なく）欠点が見つかり、内容は色あせてしまうのである。したがって、マニュアル作成の最終段階では、

どの部分が追加、削除、置換え、あるいは修正の対象となるのか検討すべきである。実際、マニュアルの保守（メンテナンス）をうまく行なうには、まずどんな種類の変更をなすべきかを知ることである。次に、読者が困惑を覚えたり、更新に付随した二重の誤りを犯さないように、変更情報をうまくユーザーに手渡し、統括管理することである。

これら5つのステップは、効果的で使いやすいマニュアルを作る上で、ぜひとも必要である。これらの最初の2つ（あるいは3つか4つ）のステップを飛ばしてしまう企業は、いい加減に設計されたコンピュータ・プログラムやマシンの機能上および構造上のありとあらゆる欠陥をそのままひきずって、マニュアルを作成することになる。

古い問題を“清算”するだけでは、マニュアル作成を効率化することはできない。マニュアル作成を最後から始める－最初の作成段階でうまくいかなかった旧マニュアルを再編集するような－方法では、まず努力はむくわれない。能率の悪いマニュアル作成過程を自動化しても意味がないように、ダメなマニュアルを際限なく手直ししても、時間と金を無駄に使うだけである。

図表 5.3 ＜マニュアル作成のデータ・フロー・ダイヤグラム＞

5.3　マニュアル作成過程の全体像
5.3.1　マニュアル作成過程におけるデータの流れ

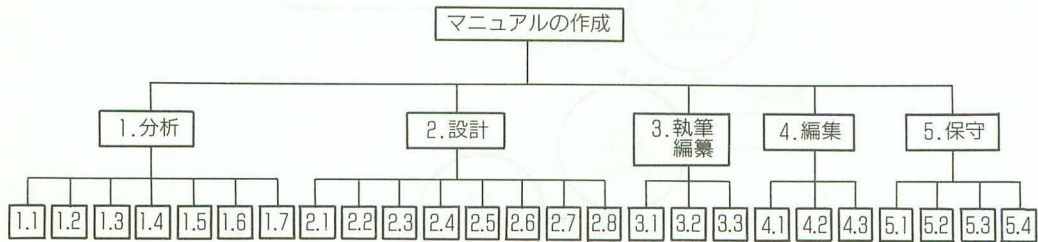
ここに示した5つの図は、前ページに示したデータ・フロー・ダイヤグラムの“子供”にあたるものである。これらは図表5.3をさらに細かく分けた1つ下のレベルの図で、5つの段階のそれぞれにおけるデータの流れを示している。（データ・フロー・ダイヤグラムでは、データは矢印の方向に動き、“丸”の中に示されたような形に変化していく。）

図表 5.3.1a　＜分析＞

図表 5.3.1b　＜設計＞

図表 5.3.1c ＜執筆・編纂＞

図表 5.3.1d ＜編集＞

図表 5.3.1e ＜保守＞

5.3 マニュアル作成過程の全体像

5.3.2 マニュアル作成過程の作業分解図

ここでは、使いやすいマニュアル、およびそれに関連する情報製品を作成する上で必要となる作業を、ピラミッド型に階層化した簡単な作業分解図を示す。

図表5.3.2bは、質の高いマニュアル作成をするために必要な作業をリストアップしたものである。また図表5.3.2aは、それらの作業を階層構造図で示したものである。これらの図は、あらゆる作業を網羅しているわけではない（たとえば、コスト評価や印刷など）が、本書の提唱する"構造化"の方法を実現するために必要な作業を網羅している。これらの作業のいくつかを省略したり、違う順序で実行しようと思うときは、それを正当化できるだけの説得力のある理由が必要である。

また、この図はそれぞれの作業を誰が実行するかについては何もいっていないことに注意していただきたい。マニュアルを書いている企業や代理業者のスタッフ構成と組織体制にはバラエティがありすぎて、誰が何をやるべきかを指定することは不可能である。その代わり、次の第6章では、各作業を詳細に取り上げ、それらに必要な技術に関して、アドバイスを行なう。

図表 5.3.2a　<マニュアル作成過程の作業分解図>

図表 5.3.2b ＜マニュアル作成過程の作業分解＞

(1.0)	**分析**：「マニュアルセット・メモ」を作る
(1.1)	システム/製品の全体像を考える
(1.2)	対象ユーザーを考える
(1.3)	機能、作業内容、項目などを考える
(1.4)	「項目/ユーザーのマトリックス」を作る
(1.5)	マニュアルおよびその他の情報製品を位置付ける
(1.6)	ばらばらな「セット・メモ」を体系化する
(1.7)	必要に応じ、プランを再検討する
(2.0)	**設計**：詳細なモデルを作る
(2.1)	従来型の(項目別の)アウトラインを書く
(2.2)	必要に応じて独立アウトライン(中間的なステップ)を書く
(2.3)	「構造化アウトライン」を書く
(2.4)	必要に応じてアウトラインを再検討する
(2.5)	構造化アウトライン中の各項目ごとに「モジュール・スペック」を書く
(2.6)	必要に応じてスペックを再検討する
(2.7)	製品のストーリーボード・モデルを組み上げる
(2.8)	モデルを最終的な段階まで見直し、テスト、改良する
(3.0)	**執筆・編纂**：初稿を作る
(3.1)	各モジュールの執筆者を決める
(3.2)	その他の分担を割りあてる
(3.3)	初稿をコーディネートする
(4.0)	**編集**：初稿を編集する
(4.1)	明確さと読みやすさのために編集する
(4.2)	技術的な正確さのために編集する
(4.3)	初稿の有用性を確認する(実際のユーザーにテストさせる)
(5.0)	**保守**：マニュアルをサポートおよび更新する
(5.1)	マニュアルおよび他の情報製品のバグを取る
(5.2)	製品のバージョンアップを検討する
(5.3)	マニュアルのバージョンアップを検討する
(5.4)	メンテナンス用の情報製品を出荷する

第6章

分析：どんなマニュアルが求められているか

6.1　マニュアルはユーザーサポートの一形態

マニュアルの中身を決める場合、そのマニュアルが含まれる文書や情報製品のさらに大きなセットを定義しておかなければ、解決できない基本的な問題がいくつかある。

●森を見てから木を見るべし

マニュアル作成を開始する際に正しい方針を立てようとするならば、そのマニュアルを含む情報製品の「全体像」（マニュアル、リファレンスカード、ビデオテープなど）を明確化すること、そして各情報製品が全体の中でどのような機能とどのような守備範囲を持つかを、明確化することが必要である。なぜなら、事柄Ｔを明確に定義するためには、事柄Ｔに含まれないことを明確に定義しなくてはならないからである。マニュアルの目的を明らかにするには、マニュアルを他の近接関係の文書と対置させるという方法が、もっとも確実である。

ある問題を解決する糸口の1つは、その問題をより大きな問題の一部と見ることである。マニュアルＡに何を書くべきかを知るためには、そもそもなぜマニュアルというものが（それも複数の）必要なのか、それら複数のマニュアルは全体としてどんな機能を果たすのか、その全体の中で個々はどのような働きをするのか、ということを知る必要がある。

実際、複数のマニュアルの全体像を明確に定義する最善の方法は、マニュアルをより大きな総体（いわゆる「情報製品」と呼ばれるもの）の一部とみなすことである。その総体には、マニュアルのみならず、オーディオビジュアル製品、オンライン・チュートリアルなどをはじめ、ユーザーの教育や情報参照を目的としたあらゆる領域のメディアが含まれる。

さらにいうならば、ユーザーにとって必要な情報製品の全体を明確化するには、ユーザーサービスの全領域を加えた、さらに大きな総体である「ユーザーサポート」を考え、この一部に情報製品があると考えるのが、もっとも適切な方法である（図表6.1参照)。

また、情報製品の必要量とユーザーサービスの必要量には、交換可能な関係があることにも注意して欲しい。情報製品が高品質であれば、トレーニング、コンサルティング、メンテナンスなどの必要性を引き下げることができる。実際、これらのユーザーサービスとマニュアルの品質とのバランスが、マニュアル作成にかかる費用の多寡を決める主要な評価基準となるのである（つまり、マニュアル作成にいくら費用をかけても、ユーザーサービスを軽減しないようなマニュアルでは、その費用は無駄ということになる)。

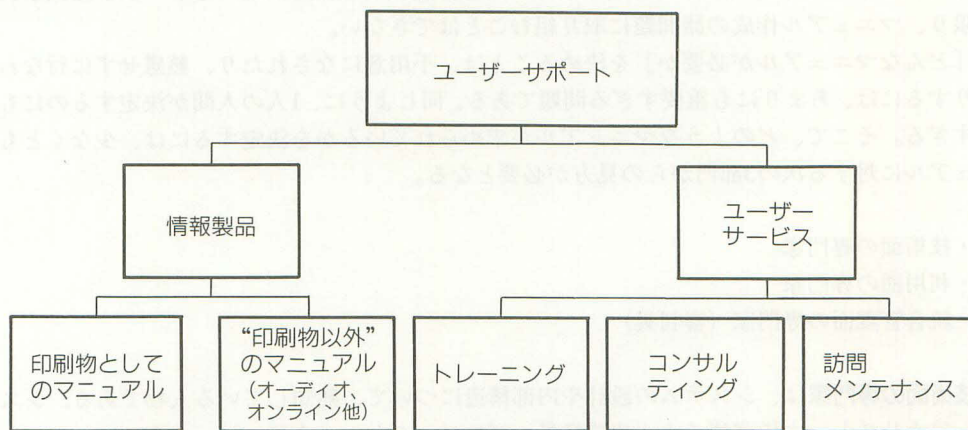
簡単にいうと、ある1つのマニュアルの守備範囲を決めるのは、アウトライン作成時でも、もちろん初稿執筆時でもなく、アウトライン作成の「前」である。また、誰がどんな情報を必要としているかについての議論は、マニュアル作成作業の最初に行なわれ、2冊のマニュアルの内容が重複しないかという議論は、双方のアウトラインが書かれる前に行なわれる。

●マニュアルの問題は書く前に発生する

こうした根本方針の重要性に気付いていないはずはないのだが、いまだにほとんどのマニュアル執筆者は、この問題に真剣に取り組もうとしない。プログラムを書きたくて仕方のないプログラマーのように、マニュアル執筆者はテキストを書くことに入れ込んでしまう。書き始める前にマニュアルの守備範囲を議論するよりも、とりあえず書き始め、後になってそのマニュアルがある特定の読者に役立つかどうかと悩んでしまうのである。

しかし、結局は読者を無視していることと同じになってしまう。書き上げられた草稿は、コーディングが終わったプログラムと同様に、口出しをはねつけるものとなってしまい、執筆者は草稿の守護者となってしまう。すでに執筆が完了し、金額が算出されたマニュアルを前にすれ

ば、ユーザーや消費者のニーズなどは、ほとんど影響力を持たないのである。明確な定義が行なわれず、誤ったコンセプトの下で作成作業の開始されたマニュアルに、今日も多くの熟達した執筆者が悪戦苦闘している。不幸なことに、これらの執筆者は、マニュアルの「内容」に問題があると思っているが、実はマニュアル作成の方針に問題があるのである。つまり、情報製品に対するプランが、策定されていないということである。

有能なライターに悪いマニュアルを書かせることほど、無益なことはない。したがって、マニュアルとは何かを明確にし、何を書くべきで、何を“書いてはならない”かを知るための唯一の方法は、情報製品を1つ1つ取り上げて、全体像における他のものとの違いをはっきりさせることである。

図表 6.1　<ユーザーサポートにおけるマニュアルの位置>

6.2 プロジェクトチームを組織する

まず最初にやらなければならないことは、マニュアル作成のプロジェクトチームを組織することである。このプロジェクトチームの任務は、簡単にいうと、対象となるシステムにどんなマニュアルが要求されているかを討議し、マニュアルセット・メモ（マニュアルの企画案と、その簡単な内容説明をリストアップしたもの）を用意することである。チームの運営を成功させるには、まず、技術関係の専門家、次にその技術を利用する側の専門家、そして両者のまとめ役としてのコーディネーターが必要である。

作成すべきマニュアルの方向を定め、その内容を説明する任務にあたるチームが組織されない限り、マニュアル作成の諸問題に取り組むことはできない。

「どんなマニュアルが必要か」を決めることは、不用意になされたり、熟慮せずに行なわれたりするには、あまりにも重要すぎる問題である。同じように、1人の人間が決定するのにも重大すぎる。そこで、どのようなマニュアルが求められているかを決定するには、少なくともマニュアルに対する次の3部門からの見方が必要となる。

・技術面の専門家
・利用面の専門家
・統合管理面の専門家（審判員）

●**技術面の専門家**は、システムの設計や内部構造について、熟知している人物である。システム・アナリスト、主任デザイナーや開発員、プロジェクト・マネジャー、あるいはエンジニア本人などがこれにあたる。こうした技術面の専門家は、システムの側に立って発言しなければならない。システムがほとんど完成している状態なら、技術面の専門家はその機能および特徴について一番よく知っている人間のはずであり、システムが開発途上の場合は、その機能仕様、あるいは全般的な設計を担当しているはずである。

●**利用面の専門家**（“ユーザー”と断定してもかまわない）は、システムのもたらす利益を知っていなければならない。ユーザー、エンドユーザー、オペレーター、工場関係者、熟練作業員、監査役、マーケティング・マネジャー、トレーニング担当者、顧客管理係などが、これにあたる。これらの人々は、誰もがプロジェクトチームの参加メンバーとして最適任といえるだろう。利用面の専門家の任務は、チームのメンバーに対して、機会あるごとに、「システムは使用され、操作されなければならない」ということを、想い起こさせることである。ユーザーやオペレーター（取締役や経営者、あるいはセールスマンも含む）とは、システム内部の働きにほとんど興味を持っていない人々、技術面の知識をほとんど、あるいはまったく持っていない人々であり、システムが何をもたらしてくれるかについて、ほとんどつねに過剰の期待を持っている人々のことである。

●**ほとんどの場合、これら2方向からの見方は衝突する。**たいていユーザー側が論争をしかけ、アナリスト側がそれに応戦するということになる。したがって、チームの第三のメンバー、すなわち**統合管理面の専門家（コーディネーター）**が意見の衝突をうまくさばき、双方から合意を引き出す役割を担う。ドキュメンター（あるいは“ドキュメンタリスト”）、“ビジネス・システム・アナリスト”、渉外担当員、品質検査員、テクニカルライター、あるいは出版技術の専門家などが、これにあたる。統合管理面の専門家は、実際的なメモやリストを作成しなければ

図表 6.2　<マニュアル作成プロジェクトチーム>

観点	メンバー	
・技術面	・システム･アナリスト ・主任デザイナー/開発員 ・プロジェクト･マネジャー ・ハードウェアの専門家 ・ソフトウェアの専門家	
・利用面	・ユーザー ・オペレーター ・工場関係者 ・熟練作業員	・監査役 ・マーケッター ・トレーニング担当者 ・顧客管理係
・統合管理面	・ドキュメンター/エディター ・ビジネス･システム･アナリスト ・渉外担当員/コーディネーター ・品質検査係 ・テクニカルライター ・出版技術の専門家/マネジャー	

ならない。彼または彼女は、双方の意見を注意深く聞き、今後話し合われる事柄について、その進むべき方向を管理する人々であり、対象となるシステムに関するマニュアルセット・メモを作成する人々である。

●メンバーのバランスは取れているか

このようなプランでマニュアル作成を行なうと、マニュアル作成の責任者は、マネジャーやコーディネーターとなることに注目して欲しい。彼らはプランニングの初期の段階から参加しているメンバーであり、乱雑な初稿を整理するコピー・エディターではないのである。また、マニュアルの必要性を定義するという仕事は、技術面の専門家、あるいは利用面の専門家のどちらか一方に任せられるものではない、ということも忘れてはならない。なぜなら、この2種類の専門家が相手の意見を尊重することなど、ほとんどありえないからである。実際の作業では、この2者間のコミュニケーションが、非常に難しい場合が多い。

理想的には、チームは3人のメンバー（各部門から1人ずつ）で構成されるべきである。システムが非常に複雑な場合、あるいは対象ユーザーが普通以上に多方面に渡っている場合などは、メンバーはもう少し多くなるだろう。しかし、注意して欲しいことは、もし技術面か利用面のどちらかの専門家が複数いると、そちらの勢力が勝ってしまう、ということである。たとえば、ハードウェアとソフトウェアの専門家が複数いる場合には“システムのもたらす利益”に立って発言する人間も複数必要である。それができない場合は、特定の技術分野の専門家がマニュアルの内容とは無関係な記述を挿入しないよう、コーディネーターはつねに注意を払わなければならない。

また2人だけのプロジェクトチームにも用心しなければならない。“能率アップ”を考えて、有用な論争を避けてしまうからである。特に用心しなければならないのは、マニュアルの必要性を定義する仕事が、たった1人の人間に任される場合である。

6.3　機能と項目をリストアップする

技術者がチームの中で果たす役割は、システム構成・機能などの「項目」をリストアップすることである。これらは、マニュアルにどうしても書かなければならない事柄である。確かに、システムのあらゆる側面を分類するには、無限の組合わせが存在するが、業務内容別の分け方が最適ではある。しかし、マニュアルの内容分析を完全で、より詳細なものにするためには、ある特定のアプローチに重きを置きすぎるべきではない。

ある製品にどんなマニュアルが必要かを分析する場合、まず最初にやらなければならない作業は、どんな製品かを明確に定めることである。いわば、その製品の商品価値を判定することである。

●分解作業のさまざまな観点

項目の分析（あるいは機能分析）は、プロジェクトチームにおけるシステムの専門家の仕事である。この分解作業は、チームの他のメンバーからも当然影響を受けるが、システム自体の構造や形態を説明するのは、やはりシステムの専門家の仕事である。

システムは、ハード構成、設計、テクノロジー、オペレーション（操作方法）、アプリケーション（適応業務）、あるいはユーザーの利益など、さまざまな観点から説明され得る。

図表6.3aは、ハード構成およびそれに付随する事柄の面から、システムを分解した一部である。このような分解作業は、システム・プランニングの一環として、すでになされていることが多い。

さらに、他のアプローチとして、アナリストやエンジニアにとってはやや畑違いかもしれないが、図表6.3bのように、販売戦略、あるいはシステムがユーザーに与える利益という観点からシステムを分解する方法もある。

●業務別マニュアルとは何か

1980年代においては、マニュアル作成班の間で「業務別マニュアル」についての討議が、かなり多くなされている。業務別とは、対象読者が実行する業務に合わせて、マニュアルが組み立てられているということである（図表6.3c参照）。この業務別（製品別と対比されることが多い）とは、技術インストラクターが“スキル分析*”と呼び、一部の社会学者たちが“作業評定*”と呼んでいるものに相当する言い方である。簡単にいうと、業務別のマニュアルとは、ユーザーの行なう業務だけをサポートし、システムの特性に関する一般的な説明を控えたものである。さらにいうと、こうした考えに関するIBMバリアント*では－ITCC*の最近の会合において提示された多くの報告書にも紹介されているが－“一般業務体系*”（考えられる限りのあらゆる業務と、あらゆるソフトウェアやシステムに関する標準分類表）を作成しようという試みがなされている。

今後さらにはっきりとした傾向になるとは思うが、マニュアルから逆戻りや回り道をなくすための最善の方法－マニュアルの有用性を高めるために－は、はっきりと定められた対象者に向けて、業務別のマニュアルを作ることである。ただし、項目の分解作業が、最初から業務別に行なわれ「なかった」としても、マニュアル作成の後の段階で、このアイデアを組み入れる機会は、まだあるのだということに留意していただきたい。

●どこまで分解すべきか

機能および項目を定義する上で、業務別という分解作業の他にも、さまざまな分け方がある。マニュアルを作成する場合、業務別の分解作業は、ハード面での分解作業より便利であること

インプット・デバイス
アウトプット・デバイス：VDT
アウトプット・デバイス：プリンター
アウトプット・デバイス：プロッター
ディスク・ストレージ
テープ・ストレージ
RAM
ROM
マイクロプロセッサー・チップ
OS
ユーティリティ

図表 6.3a ＜ハード構成別項目リスト＞

キーボードの再定義
ユーザー登録用ファンクションキー
出力形式ツール
“マクロ定義*”のできるワープロ機能

図表 6.3b ＜販売戦略／利益別項目リスト＞

製品の選択／評価
プランニング／設置準備
インストール
ハード構成登録
カスタマイジング／初期値設定
アプリケーション(適応業務)の記述
トラブル対策／メンテナンス
ファイル・コンバート
キーボード・テンプレートの作成
他のシステムとのデータ交換

図表 6.3c ＜業務別項目リスト＞

が多いが、どんな分け方を用いるにせよ、充分な分析がなされるまでは、続けられるべきである。

果たしてどこまで続けられるべきか。プロジェクトチームのメンバーが、次のような全体的な質問に答えられるようになるまで、項目は細分化されなければならない。つまり、「項目Tは読者Rにとって必要か」という質問に、「必要である」あるいは「必要でない」と明確に答えられるようになるまでである。項目の定義が、大ざっぱであったり、曖昧であったりした場合は、もう一度分析をやり直さなければならない。

原注＊スキル分析：ある修学課程や学習プログラムにどんな項目を含めるべきかを明確に定義する方法。主に学生や学習者がある作業を遂行するのに必要とする知識や能力を基準としている。
＊作業評定：人間の行なうあらゆる業務を観察し、記録する方法。考えられる限りのあらゆる行為および相互行為を書き込んだ分類表を使用して行なわれる。
＊IBM バリアント：IBM が開発中の、業務別マニュアル作成のための特別な手法。
＊ITCC：International Technical Communication Conference。アメリカの Society for Technical Communication がスポンサーになっている年1回のシンポジウム。
＊一般業務体系：IBM の開発した作業評定表で、ユーザーが自分の仕事を遂行するのにどんな文書を必要とするかを分析するのに利用される。IBM Publication Guidelines: Designing Task-Oriented Libraries for Programming Products を参照（注文番号 ZC28-2525）

訳注＊マクロ定義：長い操作手順を2～3のキー操作でできるようにする機能。

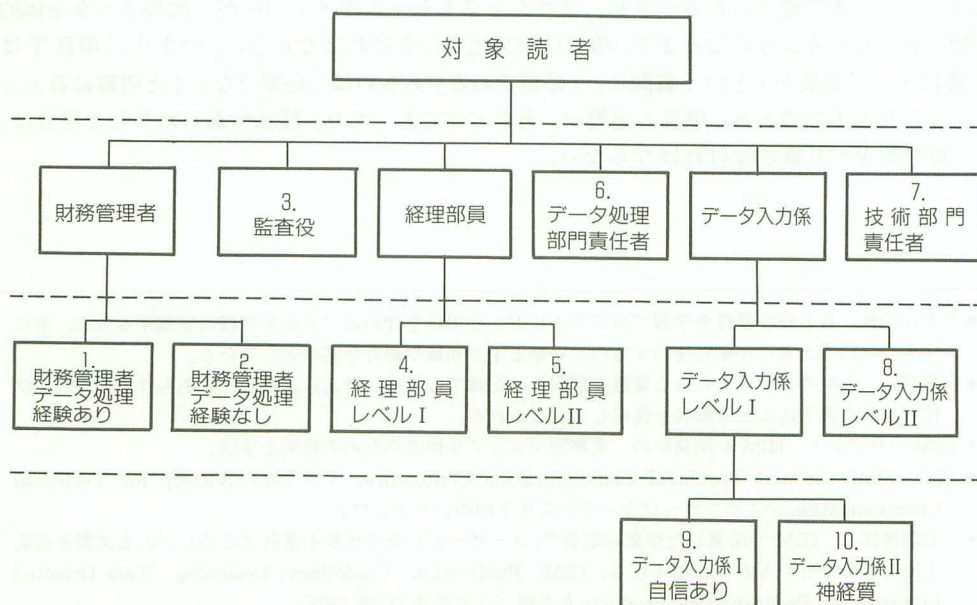
6.4 対象者を分析する：ユーザーと読者

どんなマニュアルが必要かを定義する上で中心となる課題の1つは、マニュアルの対象者（システムに関する情報を必要としているユーザーおよび読者の層）を分析することである。対象者に関する分析や階層化は、プロジェクトチームの中で、ユーザーやオペレーターにあてたマニュアルの部分を担当する者、あるいはシステムの使われ方に関する専門家としてチームに参加している者にとっての責任である。

●使用目的と経験で読者を分ける

「読者」をその使用目的と経験によって位置付けられるものとして定義しよう。2人の読者が、あるシステムに関し、共通の使用目的および共通の技術的（職業的）経験を持っていた場合、彼らは同じ読者層をなす。そのシステムに対し異なった使用目的（つまり異なった役割、異なった適応業務など）を持つ人々は、異なった読者層に属する。同様に、異なった技術的経験を持つ人々は、たとえ同じ目的でそのシステムを利用するとしても、異なった読者層としてみなされるべきである。

ユーザー層の広がりは、製品およびシステムのすそ野の広がりに対応している。そして、最新の読者層は、コンピュータ・テクノロジーに関する技術的経験をほとんど持たない人々に、ますます広がる傾向にある。

とはいっても、ユーザーおよび読者を分解する作業は、やり過ぎないことが肝心である。問題になっているユーザー層を明確化する上で、せいぜい3つから10個のカテゴリーに分けるのが普通である（図表6.4の分解図は、10個のカテゴリーに分けられた場合のよい実例である）。もちろん例外的に、読者層が10以上のカテゴリーに分けられるシステムもある。読者層は、少なく見積るよりも多く見積った方がよい。なぜなら、分析の後の段階において、余分で不必要なカテゴリーは、削除することができるからである。

図表 6.4 ＜対象読者を分析する＞

まず最初の分解は、仕事の専門分野あるいは肩書によって行なわれる（厳密にいえば、同一の経験および使用目的を持つ人々は、まったく別の肩書を持っている場合でも、同じ読者層に入る）。さらに、最初の分解で得られたカテゴリーの中で、経験の有無を基準として、さらに2回目の分解を行なう。マニュアル作成においてありがちな問題は、経験豊かなベテランと、これから練習しようという初心者の両方を対象とするマニュアルを書いてしまうことである。彼らには、同じ職務が割りあてられることもあり得るため、両者を対象とするマニュアルは、「帯に短したすきに長し」となってしまう。したがって、1回目の分解で得られた職業的カテゴリーに対して、経験のレベル、あるいは経歴の違いを基準として、さらに分解作業を行ない、サブグループ化することが賢明である。

●3つ目の基準：神経質

特に、ユーザーやオペレーターにシステムの働きを説明するためにマニュアルを作る場合など、読者を定義する上で、「神経質」という基準による第三の分解作業が必要になってくるだろう。

この神経質という基準は、たいてい経験の不足や技術的訓練を充分受けていないことと密接に結び付いている。したがって、対象者の分解作業を行なう上で、あえてそれを考慮に入れる必要はない。しかし、たとえば“高校出のキーパンチャー”と呼ばれる読者が対象となっていたら、ほとんどの読者が、システムに対し、神経質で自信がないと予想することができる。また、あるシステムをCAD(コンピュータ援用設計)用に使用している技術者グループが読者だった場合、彼らのほとんどが自信を持っており、システムやマニュアルには、むやみに神経質にならないと予想することができる。

読者の分解作業は、重要な課題である。**マニュアルを、読者全般を対象とする分かりやすい説明書として作成することは、方針上の深刻な誤りである。**この分解作業において大切なことは、オペレーター、マネジャー、プログラマーなど、システムに接しようとするあらゆるユーザーに対して、彼らが本当に欲しているマニュアルを、確実に提供できるようにすることである。さしあたりここでは、ユーザーと読者をできる限り細かく分解するという“もっとも初歩的なケース”で充分である。どんなマニュアルが求められているかについて決断を下すには、誰がなぜ、そのマニュアルを求めているのかについて、まだまだ熟考しなければならないからである。

6.5　項目／対象者のマトリックスを作成する

2つの分解作業をもとに、プロジェクトチームのマニュアル・コーディネーターは、項目／対象者のマトリックスを作成する。次にチーム全員で、マトリックスの縦と横にどういう接点があるかを分析する。この接点は、どの項目がどの対象者に対応しているかを表わしている。

●どの項目はどの読者のために？

プロジェクトチームは、項目と読者に関する2つの分解図を読者／項目のマトリックスに書き換える。そして、どの項目がどのユーザーの使用目的に合致するかを判断する。

この判断について、プロジェクトチームの意見が対立することはまれである。意見の不一致があった場合、一番簡単な打開策は、とりあえず、問題の項目もマトリックスに含めておくことである。(どんなマニュアルが必要かについての選択を行なう場合、もっとも無難なやり方は、情報を切り捨てるよりも、余分に取り入れておくことである)。

図表6.5において、各項目は「必要である」あるいは「必要でない」と明確に答えられるまで細分化されている。つまり、特定の読者がその項目の全体を知る必要があるかないかが判断できるまで、細かく分けられているのである。

おもしろいことに、この種のマトリックスは、企業の従業員研修部門で作成されているものに類似している。それは“スキルマトリックス*”あるいは“タスクマトリックス*”と呼ばれ、さまざまな従業員の訓練事項の要否を決定するのに使われている。もし、会社の研修部門で、この種の分析がすでになされていれば、マニュアルに何が必要であるかを分析する上で、それらのマトリックスが役に立つだろう。

●マトリックスから読み取れるもの

このマトリックスで、何が分かるだろうか。マトリックス作成は、プログラマーのいらいらを募らせ、プログラムの再検討を促すような手間のかかる作業のように思えるが、実はまったく逆である。

表の中でチェックマークの付けられた部分は、さまざまなマニュアルの形態を暗示している。マトリックスを作成しないと、出来の悪いマニュアルが作られてしまう可能性が高くなる。

マトリックスは、シンキング・ツール（考えるための道具）である。もし、チェックマークの付いていない項目があったら、対象者に含むべき読者をぬかしている可能性を考慮しなければならない。また、チェックマークの付いていない読者がいたら、その読者に必要な情報が抜け落ちているということである。

さらに、このマトリックスは、違った使用目的を持っていると予想される人々が、非常に似通った情報を必要としていることも示してくれる（たとえば、データ管理部門の責任者とデータ入力係については、しばしば似たようなチェック結果が現われる)。もっと大切なことは、ある読者が無視されている、不適当な情報に惑わされる可能性がある、振り回される恐れがある、まったく違った情報を必要としている読者と同一視されている、といった事柄がおのずと判明することである。

この長い項目のリストを、マニュアルの記述内容の在庫目録として考え、読者層のリストをこの記述内容の消費者と考えて欲しい。そうすれば、項目／対象者のマトリックスの最終目標とは、在庫の中から、消費者の便宜と必要性に合わせて、どのように情報を提供したらよいか、その「在庫仕訳」の方法を決定することになる。

	項目／対象者マトリックス（「生産計画システム」の場合）										
		管理部門 1	財務部門 2	生産部門 3	経理部門 4	データ入力（経理） 5	データ入力（DP） 6	技術部門 7	法対策部門 8	保安部門 9	品質管理部門 10
1	プロジェクトの目的	●	●	●						●	●
2	実行スケジュール		●	●						●	●
3	経費節約	●	●	●						●	●
4	品質向上	●	●	●							●
5	設置環境準備			●						●	●
6	安全対策			●					●	●	●
7	導入手順			●			●			●	●
8	カスタマイジング			●			●			●	●
9	ファイルの初期設定			●			●				●
10	生産プランの導入			●		●	●				●
11	毎日の作業			●		●	●				●
12	生産プランの修正			●		●	●				●
13	エラー・データのロード			●		●	●				●
14	定期報告	●	●	●	●	●	●				●
15	生産予測	●	●	●	●		●				●
16	単価分析	●	●	●	●		●				●
17	ケース別プラン	●	●	●							
18	伸び率グラフ			●	●		●	●			
19	予想グラフ			●	●		●	●			
20	コスト・グラフ			●	●		●	●			
21	会計管理		●	●	●				●		●
22	契約管理		●	●	●				●		●
23	データ入力エラー			●		●	●			●	●
24	エラー報告			●		●	●			●	●
25	操作手順不良			●			●			●	●
26	プログラム一覧			●			●			●	
27	データ項目一覧			●			●		●	●	●
28											
29											
30											
31											

図表 6.5　<項目／対象者のマトリックス>

原注＊スキルマトリックス：修得すべき技能を縦にし、教育の対象となる学習生のグループを横にした表。

＊タスクマトリックス：説明すべき（マニュアルに記述すべき）業務を縦にし、ユーザーや得意先のグループを横にした表。

6.6　分化：境界部分と重複部分を分ける

プロジェクトチームは次に、項目と対象者のマトリックスを検討して、そこから、いくつかのパターンやグループを洗い出し、それによってマニュアルを区分けする。分化された一方の極においては、1冊に収まった事典的な汎用型のリファレンス・マニュアルが考えられる。その際、マトリックスの項目は、そのまま読者のためのインデックス（索引）になり得る。またもう一方の極では、各対象者あるいは項目ごとに、いくつかの分冊に分けられたマニュアルが考えられる。この両極間のどこかに解答があるはずだ。つまり、マニュアルの理想的な分化は、下図に示したような、双方の折衷案によって決定することができる。

システムの中に境界線をひくのと同様、マニュアルに境界を定める場合にも、多くの決定とトレードオフが－たいてい独断的に－なされる。

プロジェクトチームのメンバーが、マニュアル、オーディオビジュアルおよび他の情報製品のもっともコストパフォーマンスのよい組合わせを考える場合、さまざまな要因が互いにせめぎ合う。もちろん、企画したもの以外の情報製品を作る会社はほとんどないので、マニュアル全体は、汎用型（百科事典型）の極に傾く傾向がある。

●対象者別マニュアル

図表6.6に見られるように、2つの方針には、それに有利に働く特徴と不利に働く特徴がある。対象者別のマニュアルには、意思伝達の上で多くのメリットがある。10以上の読者層がそれぞれ選択できるマニュアルのバージョンがあるといっても決して過言ではない。それは結局、読者の使用目的にぴったり合致したマニュアルに行きつくことになり、他のマニュアルへの参照や迂回の道からわれわれを解放してくれる。この方針では、マニュアルは薄めになり、オーナーシップ－全員が同じバージョンのマニュアルを持っていたら持ち得ない感情－に裏付けられた威信を持つことさえできる。

	⇦対象者別マニュアル	汎用型マニュアル⇨
利点	・ユーザー特注 ・理解しやすい、簡潔 ・個々のユーザーに親しみやすい ・安全性を確保する ・濫用を避ける ・トレーニング・マニュアルの役割を果たす ・経費を節約できる（場合に応じて）	・効率的、冗長的ではない ・プランニングを簡単にする ・製作作業を簡単にする ・メンテナンスを簡単にする ・技能習得にも使える ・ユーザー間のライバル意識を排除する ・よいオンライン・システムをもたらす
欠点	・メンテナンスを複雑にする ・ユーザーを過小評価する ・技能習得を妨げる ・経費がかかる（たいていの場合）	・厚く重い本 ・ユーザーに圧迫感を与える ・トレーニング担当者やサービス要員に重荷となる ・複雑な索引を必要とする

図表 6.6　<マニュアルをタイプ分けした場合の両極>

各対象者にあてられた薄いマニュアルは、またある種の読者が手に取るのを制限することによって、安全性を確保することにもなる。同じように、オペレーターやユーザーが、使い方のまだよく分かっていない操作や機能を試そうとするのを防ぐ役目も果たしてくれる。場合によっては、さまざまなバージョンのマニュアルを作ることは安くつくことさえある。われわれはときどき、薄いマニュアルのコピーを100部あるいは、1000部取る必要に迫られるが、厚いマニュアルをコピーする必要はあまり出てこない。

しかしながら、対象者別のマニュアル作りはたいていの場合高くつき、メンテナンスも非常に難しくなるだろう。明らかに、1つのバージョンだけ（そのマニュアルの他のもろもろのバージョンは放っておいて）バージョンアップを継続的に行なう、ということは難しい。対象者別のバージョンはまた、対象となる読者の能力を過小評価したり、彼らの実力を向上させるような技能の習得を妨げたり、他のさまざまなマニュアルを参照させるような事態も引き起こす。

●汎用型マニュアル

ほとんどのマニュアル作成者は、作ろうとするマニュアルのニーズを対象者別で分析しようとはしない。むしろ彼らは、マニュアルを「1つのまとまり」、著述およびデータの1ファイルと考えている。このような汎用型のマニュアルを選ぶことが正しい場合もある。つまり、同一の層に属するユーザー向けの簡単なシステムの場合には、1冊のマニュアルが一番よいだろう。しかし、マニュアルを1冊にすることは、読者にとっての使いやすさをあまり考慮せずに、マニュアルの作成プランを単純化し、短期コストを低下させるための便宜上の方法であることが多い。しかも、汎用的なマニュアルでまかなおうとする会社の多くは、インデックスを付けることを怠りがちである。百科事典に、索引もクロスリファレンスの方法も用意されていなかったとしたら、読者がそれを使用することはほとんど不可能である。かといって、それがすぐに複数のマニュアルの方がよいということにはつながらない。

マニュアルを作成する上でのもっとも興味深いパラドクスは、事典のような汎用型マニュアルー大多数のユーザーにとって、もっとも厄介でなじみにくいーが、マニュアルのオンライン・システムにとっては、理想的であるということだ。難しい検索ルーチン、クロスリファレンスや回り道、不必要な部分のトリミング、その他もろもろの処理をしなければならないのは、ユーザーではなく、システムの方である。検索プログラムは、読者にとっては“透過的*”である。これは、コンピュータ用語でいうところの“不可視的”に等しい。

訳注＊透過的：transparent。通常の英語では「透明な、明白な」という意味だが、ここでは中身が見透せるという意味ではなく、ユーザーが知らず知らずのうちに検索プログラムに入り、知らず知らずのうちに出ることができる、つまりユーザーが“意識しなくともよいもの”という意味で使われていると思われる。

6.7　マニュアルセット・メモから

プロジェクトチームの討議の結果をもとに、「マニュアル・コーディネーター」は、上層部の承認を受けたプランに関するメモを作成する。そのプランには、マニュアルおよび他に開発の予定されている情報製品の概略が盛り込まれている。

プロジェクトチームは、マニュアルセット・メモを作成し、配布する。このメモには、システムのライフサイクルの最後（システムがお払い箱になる時点）までに書かれるあらゆる文書に関するプランが盛り込まれる。言い換えれば、長い射程で見た標準的なマニュアル・セットの中から、現在進行中のプロジェクトにあてはまるマニュアルを選ぶわけである。

マニュアルやその他の情報製品に関し、次のようなことがメモに書き込まれる：

●**表　　題**－マニュアルの詳しい名称。そのマニュアルの守備範囲と対象ユーザーが分かるような表現にする。

●**コード番号**－関係者全員が、マニュアルの進行とコストを管理できるように付けられたコード番号。

●**メディア**－情報の伝達形態。そのマニュアルは書籍なのか、もしそうなら形式と材質はどのようなものか（たとえば、ルーズリーフ・バインダーなど)。さもなくば、書籍以外のメディアが使用されているのか（たとえば、ビデオ、ポスター、プレースマットなど)。

●**読　　者**－対象となる読者の職業上のカテゴリーに関する簡単な記述。分かっている場合には、特定の組織、団体、個人の名称も。

●**優先順位**－同一セット内のすべてのマニュアルに関する相対的な重要度の目安。特に、同一セット内の各マニュアルに割りあてられた人員や資材が不足しそうな場合などには、重要な役割を果たす。

●**完 成 日**－マニュアルの完成予定日。プロジェクトが大きい場合には、いくつかのマイルストーン*でもよい。たとえば、ストーリーボード、第一稿、“実際のユーザー”によるテスト、最終デザイン、印刷など。

●**責 任 者**－マニュアル作成を予定通りに、仕様書通りに、そして予算の範囲内で進めるべき人。この責任者が決定するまで、マニュアルは書き始められるべきではない。“ライター”の肩書は、責任者の仕事を意味しないことに注意すること。実際、ライターの1人が責任者になることは避けるべきである。

●**予　　算**－プロジェクトにあてられた人件費、その他の予算に関する記述。明細は必要に応じて。マニュアル作成は確かに高くつくが、システム開発の他の側面と一緒に、開発コストとして“ひとまとめ”にしてしまうのには無理がある。

図表6.7は、項目の書き込まれたマニュアルセット・メモの一部である。これは、分析作業の段階のアウトプットであり、設計作業の段階へのインプットである。**どのようなマニュアルが要求されているのかについて、充分検討した後でなければ、マニュアルのアウトラインを書くべきではない。**

表題：生産計画システム用明解ユーザー・ガイド

メディア：ルーズリーフ　　コード番号：ＰＳ－０１　優先順位：Ａ

読者：生産部門；品質管理部門；保安部門

責任者：ヘミングウェイ　　完成日：’８５．３

セクション／内容
プラン、利益、準備／導入、データ入力、報告書、分析、
グラフ化、エラーとプログラムのサービス、リファレンス・ツール

予算：３０，０００ドル（人件費）；７，０００ドル（製作費）

表題：生産計画システム用オペレーション・マニュアル

メディア：ルーズリーフ　　コード番号：ＰＳ－０２　優先順位：Ａ

読者：データ入力係（経験者）

責任者：スタインベック　　完成日：’８５．３

セクション／内容
導入、カスタマイジング、初期設定、データ入力、レポート作成、
グラフ作成、エラー、リファレンス

予算：９，０００ドル（人件費）；２，０００ドル（製作費）

表題：生産計画システムを使った応用例題集

メディア：スパイラルバインダー　コード番号：ＰＳ－０３　優先順位：Ｂ

読者：技術部門

責任者：カンディンスキー　　完成日：’８５．５

セクション／内容
伸び率グラフ、予想グラフ、コスト・グラフ

予算：１，５００，００ドル（人件費）；５００ドル（製作費）

図表 6.7　＜マニュアルセット・メモの記入例＞

訳注＊マイルストーン：milestone。プロジェクト・マネジネントの用語で、期間（日数）がゼロの作業。プロジェクトの進行状況を監視するチェックポイントの役目を果たす。もともとは1マイルごとに置かれた里程標を示す。

第7章 設計Ⅰ：アウトラインを構造化する

7.1 従来型のアウトライン：その功罪

従来型のアウトラインは、複雑な要素をまとめるのに2階層の列挙を行なっている。従来型のアウトラインは、各項目の章立てと、それらと階層構造をなすサブ項目群を示している。残念ながら、従来型のアウトラインは、作業プランの中で要求されるごくわずかの情報しか与えてくれない。たとえば各セクションやマニュアル全体の分量や大きさが指定されているわけではなく、またマニュアルの制作費に関する手がかりを与えてくれるわけでもない。さらにそれは、マニュアルの中身を表示する目次としても、読者がどこを読むべきなのかの手助けにはならない。

●従来型アウトラインの利点

従来型のアウトラインは、テキストの書かれる筋道や階層構造を表わしたものである。このアウトラインを作成するには数字同士、あるいは数字と文字の組合わせを縦に並べ、下層階層を表わす（図表7.1)。そして、これに文書の各セクションの中身や趣旨を表わす表題（ヘディング）を付ける。たいていのヘディングは、名詞と形容詞のみである。

これら従来型のアウトラインは、技術文書のプランニングにおけるもっとも一般的で便利なツールである。しかし、モジュール化されたマニュアルを開発するためには、それなりの設計が必要となるが、従来型のアウトラインでは、そうした設計作業を完全に行なえるだろうか。従来型のアウトラインの主な利点は、ライターが自分の考えをまとめるのに役立つという点である。したがって、1人で仕事をする技術者には理想的なプランニング・ツールである。

●従来型アウトラインの欠点

従来型のアウトラインは、マニュアルの設計者が、マニュアルの分量や、そのマニュアルに必要な人員や資材などを見積る上で役立つだろうか。従来型のアウトラインは、さまざまなヘディングの後に続くテキストを書かなければならない人々に、役に立つアドバイスを与えてくれるだろうか。従来型のアウトラインは、マニュアルの中身に関する便利な一覧表となり得るだろうか。

ライターが1人で、その書く分量も比較的少なければ、従来型のアウトラインで充分な場合が多い。しかし、ライターがチームとして構成され、マニュアルの分量が多くて複雑な場合には、従来型のアウトラインでは不充分である。それだけでは、フリーのライターには、書くべき量が分からない。番号が付けられた概略や通常のやり方で書かれたヘディング（動詞的な要素やテーマの含まれていないもの）だけでは、マニュアルの各セクションをどのくらいの分量にすべきか、あるいはどの範囲までをカバーすべきか、などをライターが判断することはできない。

マニュアル作成に責任のあるマネジャーやアナリストは、従来型のアウトラインからほとんどデータを読み取ることができない。マニュアルの分量、図表の数やタイプなどの情報はもとより、もっとも重要なコスト面での予想はどこからも汲み取れない。マニュアル作成においては、図や表がコストの75%～80%以上を占める場合もあり得るのである。

●本には階層がない

結局のところ、この舌足らずな作業プランが、よく分からない内容のマニュアルを産み出している。しかも読者は、アウトライン中の階層構造がどうなっているかを理解しなければならないことになっている。ところが、実際にはマニュアルは単一階層であるため、2階層のアウトラインで読者を案内するのは無駄な努力である。

4.0　ライブラリアン機能
4.1　ライブラリアン
4.2　コア・イメージ・ライブラリ
　4.2.1　プログラムのリストアップと検索
　4.2.2　段階ーコア・イメージ・ライブラリ
4.3　ライブラリの再配置
　4.3.1　メンテナンス機能
　　4.3.1.1　リロケータブル（再配置可能）
　　　　　　モジュールのリストアップ
　　4.3.1.2　ライブラリ

管理者の疑問
分量は？ 図表の数は？ コストは？

執筆者の疑問
どのくらい書けばよいか？ 何を強調すべきか？

読者の疑問
どこを読むべきか？ 読み終わったのか？

図表 7.1　＜従来型アウトラインの欠陥＞

この最後の点は、確かに難しいので説明が必要だろう。本が単一階層であるか否かといった問題を考える場合には、読者がどう理解するかということではなく、「読者がどのような順番で読むか」ということがポイントとなる。ある1つの段落が「論理的」には、他の段落の下層構造をなしていたとしても、実際には「書かれている順番」に読まれるのである。事実上、本には階層構造がない。あるいは文字ではなく一連の画面だとしても、そこには「順番」がある。項目2は、実際には項目1の下でも、横でも、後ろでもない。それは項目1の「次」である。

人間である読者は、順番に読む（並行してではなく）という意味ではコンピュータに似ている。しかし人間である読者は、円環や回り道やGOTOなどの迷路から、何らかの順番を見つけ出すことは苦手である。重要な点は、従来型のアウトラインは、ライター個人の考えをまとめるのには便利だが、便利なマニュアルを企画・設計するツールとしては、不向きであるということだ。

7.2　アウトラインを構造化するために

マニュアルのみならず、ビデオ・プログラムあるいは一連のヘルプ画面を企画する者は、誰でも従来型のアウトラインを作成するところから始めるだろう。それは2段階の階層構造をなし、各章には形容詞あるいは名詞句のタイトルが付けられることになるだろう。そこで次のステップは、このアウトラインを構造化（モジュール化）されたアウトラインに変えることである。この構造化されたアウトラインでは、各章のヘディングは、ある基準に見合ったサイズとレイアウトのモジュール1つ1つに対応し、ヘディングに使われる言葉は、その章で語られる内容がよく分かるものになっている。つまり、ヘディングが「ヘッドライン」に昇華するわけである。

●構造化アウトラインの2つの条件

自分の考えを、一定基準のサイズに適合させることのできる人間はほとんどいない。したがって、作成されるアウトラインは、長さも複雑さも一定していない概念を反映している。アウトライン中の項目2.1が、項目2.2と同じ長さのセクションであると思い込む根拠はどこにもない。また、項目2.2が項目2.2.1より長いか短いかを知る術もない。

アウトライン中のヘディングも、ほとんど役には立たない。項目2.1が「管理上の補助」と呼ばれ、2.1.1が「サブシステムのアクセス」と称されている、ということが分かったとしても、問題の解決にはならない。

マニュアル作成を構造化された手法で行なうためには、この設計作業に役立つようなアウトラインが必要となるが、これには従来型のアウトラインに欠けている次の2つの要素が必要である。

- ・ライター、校正者、そして最終的には読者に、各セクションに実際には何が書かれているかを示すような文体。
- ・アウトライン中の各記載事項を、ある一定サイズの内容の“切身”に対応させるための基準。

●ヘッドラインのさまざまなスタイル

以上2つの要素のうち、前者は後者に比べると、比較的表面的な問題である。有能なライターは、たいてい従来型のヘディング（表題）よりもヘッドライン*をーアウトラインを作成しない場合は、内容一覧（目次）の中でー使用する。何十年も前から、マニュアル・ライターの多くは“勘定科目コードの振付け”（名詞3つ）、というような表現を避け、“勘定科目コードを振り付ける”“勘定科目コードの振付け方”“勘定科目コードの振付けのための6つのルール”“なぜ勘定科目コードを振り付けるのか”などの表現を使ってきた。主題を明確に提示するとともに、読者の関心を引き出すこれらのヘディング（ヘッドラインといってもよい）は、テクニカルライティングが産声を上げて以来、有能なライターの商売道具の1つとなっている。

●記載事項のサイズを統一せよ

もう1つの要素ーアウトライン中の各記載事項が、ある一定サイズの項目に対応しているということーは、ほとんどのライターにとって、なじみの薄いことである。事実、多くのアナリストやテクニカルライターは、この考えを聞いてびっくりする。量的な変動の激しい概念を、一定サイズのユニットにどうしたらまとめられるのか。そんなことが本当に可能なのか、膨大な

努力の結果なのかと彼らは思うだろう。

しっかり把握しなければならないことは、マニュアルに載せるあらゆる情報は、必ず一定基準のユニットに統一しなければならない、ということである。もっとはっきりいえば、マニュアルをはじめ、どんな出版物でも一定サイズのページで編纂され、どんなに流動的な“アイデア”でも、1ページの区画内に収められる。しかしながら、ライターやエディターの多くは、このパッケージ化のチャンスを見送っている。彼らは、1つのアイデアに何ページくらいが必要かをほとんど知らない。彼らは、自分の“申し述べたいこと”がどのくらいの長さになるかを予想することができない。実際、彼らはページの切れ目をどこにするかを、タイピストやプリンターやワープロ任せにしている。

換言すれば、マニュアル設計を構造化する方法においては、読者／ユーザーが目にすることになる実際の対象、つまり出来上がったマニュアルのページや画面を設計することが最終目標になる。マニュアルがページごとに編纂されているのに、その作成計画や設計が、なぜページごとに行なわれないのか。もし各セクション、あるいはユニットを同じサイズにすることが不可能に思えるなら、それらの取り得る上限をまず定めて、すべてをなぜそのサイズに統一しないのか。

もちろん従来型のアウトラインは、マニュアルの各モジュールについての仕様書リストに書き換えられるべきである。

図表 7.2　＜従来型のアウトラインと構造化アウトラインの機能の違い＞

従来型のアウトライン	構造化アウトライン
・記載事項が読み取れず、書いた本人にしか理解できない	・記載事項が実質的、有益かつ明瞭で、見直しやテストができる
・記載事項がマニュアルの特定の長さやサイズに対応していない	・それぞれの記載事項が一定の長さの物理的基準に対応している
・編集作業や最終稿の作成段階にならないと、マニュアルの実際の出来ばえが分からない	・マニュアルの出来上がりの形式やレイアウトは、アウトラインにもともと含まれている（モジュールがすでに定義されている場合）
・目的のマニュアルの守備範囲と制作費が、アウトラインからは伺えない	・アウトラインがそのまま作業プランになり、そこから制作に必要な経費およびその他の条件が簡単に割り出せる

訳注＊ヘッドライン：本書の著者は、ヘディングとヘッドライン、および従来型のアウトラインと構造化されたアウトラインを明確に使い分けている。ヘディングとは、構造化されていない従来型のアウトラインの１つ１つの中身に付けられる表題であり、ヘッドラインとは、構造化されたアウトラインの１つ１つのモジュールに付けられる表題である。

7.3　マニュアルのモジュールを定義する

マニュアルのモジュールは、さまざまな形を取り得る。ただし、各モジュールは、他とはっきり区別できるまで細分化され、単一化されたもので、しかも1つの機能、あるいはテーマに関して書かれたものでなければならない。また各モジュールは、「一覧できる」、つまりそのモジュールの全内容が、ページをめくらなくても見渡せるぐらいの分量でなければならない。したがって、各モジュールは標準サイズの1ページか、あるいはそれに相当する他のサイズの1ページ、せいぜい見開き1ページ（2ページ分）に収まる程度である。

●モジュール化とは何か

モジュール化されたマニュアルとは、技術的な伝達事項を単一目的の、異質な要素のいっさい含まれない状態にまで、小さく区切った切身が集まったものであり、この切身のサイズ、内容、体裁があらかじめ定められているものをいう。モジュールの設計－つまり正確な筋道－が確定したら、分量が多く、入り組んだ1冊のマニュアルを、小さいほぼ独立したマニュアルの集まりとみなすことができるようになる。小型の独立したマニュアルは、それぞれ1～2時間もあれば書け、順番を問わず、他のマニュアルとは関係なく開発することができる。マニュアル作成をモジュール化することによって、各モジュールがばらばらの順番で書かれた場合でも、出来上がったときのページと体裁を見積ることもできる。

●単純な1ページモジュール

モジュールのもっとも単純な概念は、「1ページ」である。しかし、1ページのリミットは、ほかの刊行物では理想的かもしれないが、マニュアルにおいては、ほとんどの場合やっかいなものである。細かくて乱雑な字体でページを埋めつくすことなく、あらゆる概念（それがどんなに単純なものでも）を表現するのには短すぎる。そして、図や表やテキストを同じページに同居させるには、“コピーフィッティング*”と、他の印刷技術が要求される。さらに、技術が進歩しているとはいえ、図や表の挿入されているページは、ワープロではほとんどプリントアウトできない。

●2ページモジュールの利点

「2ページ（見開き1ページ）」にまたがるモジュールは、1ページよりもメンテナンスが難しいが、応用的な使い方が期待できる。2ページモジュールの理想的な構成（すでに述べたように20年ほど前、Hughes Aircraft Corporation によって開発されたもの）は、図表7.3に示したようなものである。

A.コントロール・バー：整理番号、ページ、その他の管理データを含んだもの
B.セクション名
C.ヘッドライン（そのモジュールのテーマや機能を盛り込んだ表題）
D.要約（あるいは論題）：モジュールの中心概念を展開したもの
E.テキスト全体（通常500字＜英文で200ワード＞～2000字＜同じく700ワード＞）：ヘッドラインと要約で示された概念を展開したもの
F.図表：画面、ダイヤグラム、表、図画など－右ページに記載

見開き1ページのモジュールは、マニュアルを開くたびに－望むと望まざるとにかかわらず－人が目にするものである、ということに留意して欲しい。

A. コントロール・バー

B. セクション

C. ヘッドライン

D. 要約

E. テキスト
（500～2000字）

F. 図表

図表 7.3 ＜大型モジュール＞

あるモジュールは1ページ、11インチ×15インチのコンピュータ用紙1枚に収まることもある。あるいは、2ページモジュールの各部分の配列を変えることもあり得る。マニュアル作成者の中には、図や表を左ページに持ってくるのを好む者もいる。変化に富んでいる方がよいということで、右と左を置き換える者もいる。

しかし、これらの代替案を表示する前に、次の2つのことを強調しておかねばならない。

- モジュールの定義は厳格で、不動のものであるが、実際にはかなり柔軟で、モジュール内の単語や図表の数量に対して、かなりの許容度がある。特に、2ページモジュールでは、モジュールの実際の"重量（密度）"あるいは容積は、表面上は"同じサイズ"でも、様々に変わり得るものである。
- 特に2ページの形式は20年以上前から、官公庁やビジネスの分野の生産技術面の刊行物に、さまざまな方法で適用され、成功を収めている。つまりこの形式は、読みやすく非常に効果的ということだ。

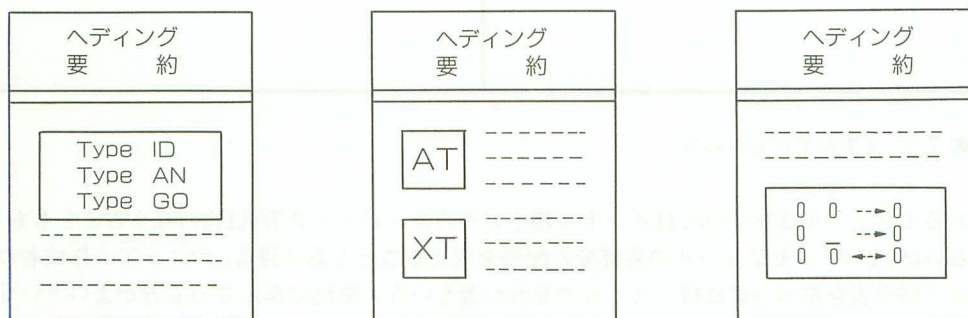
この基本的な形式は、必要に応じていくらでも変更できる。たとえば、2ページモジュールでは、テキストと挿画の配列を変えることができる。右ページが全部余白になるということさえあり得る。

原注＊コピーフィッティング：最終的な印刷に回す前の原稿をレイアウトする方法。特に図や表を最適の場所にレイアウトする作業。

7.4　モジュールのさまざまな形：特定のニーズに応えて

すでに紹介したように、見開き1ページに収められたモジュールは、ほとんどのマニュアルに通用する便利なものである。ページをめくらずに内容を一覧でき、1つの機能、あるいは概念を盛り込むには充分な大きさである。また、構造化マニュアルを考える上での最小単位を構成するに足る詳細さを兼ね備えている。1ページモジュールの例、および個々のニーズに対応する2ページ（見開き1ページ）モジュールの例などを以下に紹介する。

●1ページモジュールはマニュアルを複雑にする？

基本的なモジュールは、1ページである。図表7.4aは、この1ページの構成案を示すと同時に、この1ページモジュールの構造上の欠陥も例証している。つまり、テキストと図表を1ページにまとめるという欠陥である。このまとめ方で問題が発生しないならば、図表の形式がよほどよいか、使用した印刷の手法が洗練されているためである。この場合は、1ページモジュールが理想的となることもある。

図表 7.4a　<1ページモジュール>

1ページモジュールが技術的に可能な場合でも、ソフト開発の場合と同じように、モジュールが小さければ小さいほど、モジュール間の連結関係は複雑になるということを忘れてはいけない。つまり、それだけ参照、逆戻り、読飛ばし、回り道などの回数が増えるのである。非常に小さいモジュールでは、繰り返し書くことができない。したがって、読者を他のモジュールへ送り出さなければならなくなる。

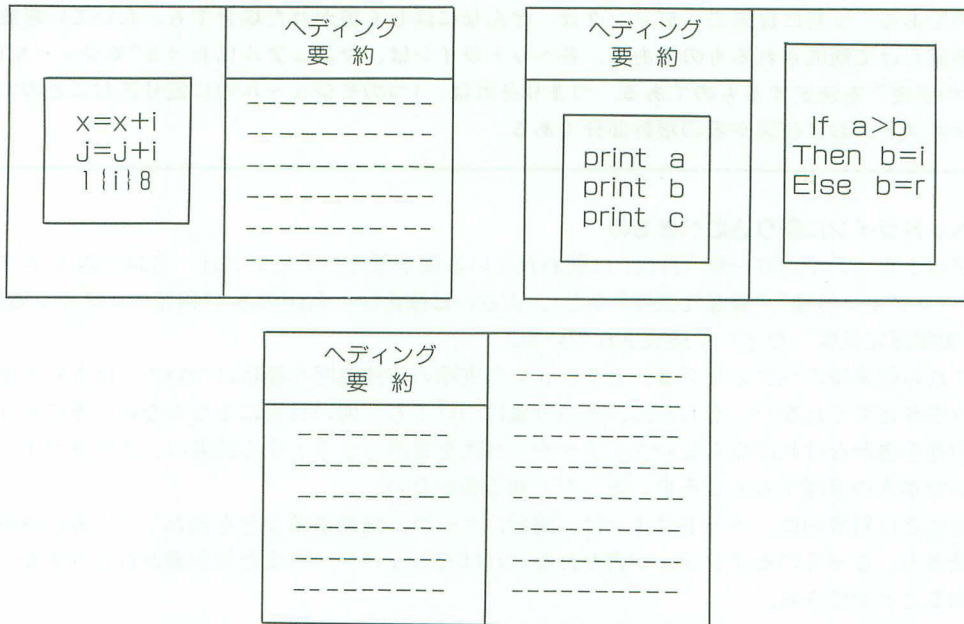
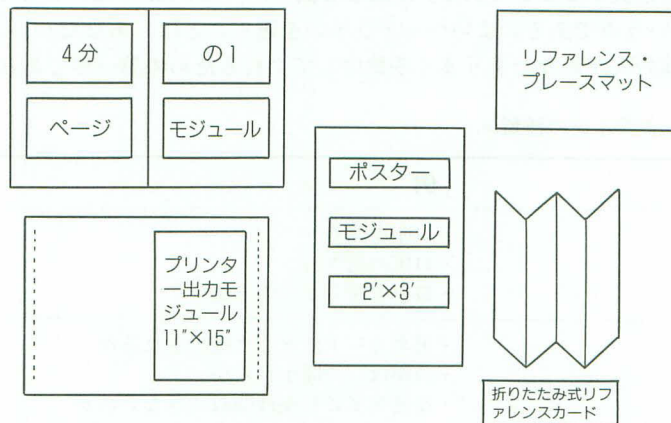
●2ページモジュールはレイアウトが自由

2ページモジュールを選んだ場合には、図表7.4bのようにモジュール内の構成要素を配置換えできる可能性も、同時に選んだことになる。

モジュールはテキスト半分、図表半分で構成されるという仮定から出発するが、テキストと図表の比率を変えた方がよくなる場合も多々ある。モジュールのほとんどがテキストで埋められるという場合もあるが、このようなモジュールが頻繁に登場するのは、ほめられたことではない。モジュールのほとんどが図や一覧表などで構成されることもそこそこある。図表を左に置き換えたり、折込みサイズにまで広げたりすることもあり得るだろう。

●その他のモジュール・サイズ

図表7.4cは、変わったサイズのページの使用例を示している。ハーフサイズのページ、ある

いはそれ以下のサイズ、11インチ×15インチのプリンタ用紙（24列×120行）、2フィート×3フィートのポスター、リファレンス・プレースマット、折りたたみ式リファレンスカードなどがそれである。

図表 7.4b　<2ページモジュール>

図表 7.4c　<その他のモジュール>

モジュールは、表現したい概念の全体を伝えるに小さすぎず、全体的なプランや概要を簡単に収めるに大きすぎないものでありさえすれば、考えられる限りのあらゆるサイズ、形態を取り得る。

したがって理想的なモジュールとは、スクロール（画面移動）しないで読める範囲－すなわち、1画面で構成されるものであろう。この場合、モジュールのリミットとサイズは、開発者の持つビデオ・ディスプレイのサイズによって決定されることになろう。

7.5 モジュールにヘッドラインを付ける

各モジュールは、そのサイズや形状にかかわらず、「ヘッドライン」によって推し進められる。ヘッドラインとは、従来のヘディングとは違い、テーマ、概念、主張、さらに論旨まで含めたものである。反対に従来のヘディングは、どんなに詳しく書かれた場合でも、たいてい修飾語と名詞だけで構成されるものである。各ヘッドラインは、マニュアルにおける“モジュール1つ1つの価値”を決定するものである。つまりそれは、1つのモジュール内に盛り込むことのできるテキスト、および図や表の根幹部分である。

●ヘッドラインに盛り込むべきもの

アウトラインや内容一覧（目次）に使われている従来型のヘディングは、名詞のみ（“ログオン”“ログオン処理”“電源冗長度”など）、あるいは修飾語と名詞のみ（“相互のログオン処理”“多重電源冗長度”など）で構成されている。

これら従来型のヘディングは、セクションの実際の守備範囲や趣旨について、ほとんど手がかりを与えてくれない。もちろん、その分量についても、何の目安にもならない。セクションの中身を書かなければならないライターや、目次を参照しようとする読者は、アウトラインを書いた本人の意図するところを、まったく知る術がない。

それとは対象的に、ヘッドラインは、論旨、テーマ、強調事項などを表わしている。執筆者も読者も、なぜそのセクションが書かれなければならないのか、また何が書かれるべきなのかを知ることができる。

効果的なアウトラインを書くためのキーポイントは、**そのモジュールで語られるべきことは何なのかを正確に把握すること**である。注意して欲しいのは、「語りたいことではなく、語られるべきこと」だという点である。よいヘッドラインを書くことは、あなたが1人で書くマニュアルでない限り、他の人があなたをうまく手助けしてくれるための第一歩であるといえよう。

図表 7.5a　<ヘッドラインの種類>

スタイル	例
動詞的	・口座を開く ・口座の開き方 ・口座を開く２つの方法
副詞的	・署名カードはどこに提出されるか ・口座はいつ開かれるか ・なぜＸＹＺ口座は開設できないのか
並列的	・新規口座：承認するのは誰か ・口座番号：どうやってそれを解読するか ・書式99：身分証明の必要性
命令的	・新しい口座の必要性 ・加入者リスト更新の重要性 ・口座振替を確認しない危険性
平叙文的	・１時間に10件の口座が処理されます ・コード番号を覚えてください ・月間報告書は受け取りましたか

●中間ステップ：独立アウトライン

ヘッドラインを書くコツは、見方を変えれば、マニュアルを一定サイズのモジュールに構造化するコツとは異なる、ということに注意して欲しい。ヘッドラインを書く場合、目的のモジュールの中身の重量や容量と切り離して考えることができるのである。事実、構造化されたアウトラインを作成するために、“独立アウトライン”という中間的な段階を設けるという方法がある。これはモジュールを考慮せずに、ヘッドラインのスタイルに書き直されたヘディングのことである。この方法（後に詳述）では、独立アウトラインが構造化アウトラインへと、さらに練り直されていくのである。

中身の分かる文体にすることと、モジュールに細分化すること－これら2つの問題を同時に解決することは、やっかいなことのように思われる。しかしやってみれば、この方法が意外に簡単であることが分かるだろう。モジュールのサイズを知ることは、結局ヘッドラインをより的確で、明快なものにすることになる。

図表7.5bは、私個人のマニュアルのライブラリから取ってきた従来型のヘディングの例である。そして右側を見れば、執筆者が何を意図して、そのヘディングを書いたかが分かるはずである（余白の部分には、あなたが自分で何か適当なヘッドラインを考えてみて欲しい)。

図表 7.5b ＜ヘディングの書き換え例＞

変更前	変更後
アクセスの仕方	2つのアクセス形式：シーケンシャルとダイレクト
ファイル	システムによるファイル認知：ファイル名と属性
エグゼクティブ・ライブラリ	タイムシェアリング・システムは、標準および特殊処理ルーチンを供給する
プログラムのデバッグ	デバッグ機能の使い方
FDEBUGの例	例題：FDEBUGでフォートランのプログラムをデバッグする
ネームリスト機能	＿＿＿＿＿＿＿＿
LOAD/USE コマンド	＿＿＿＿＿＿＿＿
透過的書込み*	＿＿＿＿＿＿＿＿

原注＊透過的書込み：transparent write。データをファイルに自動的に（ユーザーやオペレーターに意識させずに）書き込む方法。

7.6　実証：ヘディングからヘッドラインへ

この項で示される図表は、従来型のアウトラインが、構造化アウトライン、ヘッドラインのリストに変更された場合、どうなるかを表わしている。ヘッドラインの文体は、軽快で会話的な(一種の“ショッピング”スタイル)あるいは直截的で具体的なものにすることをお忘れなく。

図表7.6aでは、ほとんどのマニュアルに書かれているシステム導入プランを例に挙げ、新旧2つのアウトラインの形式を示している。“旧”バージョンのアウトラインは、データ処理(DP)部門の典型的な書式である。しかし“新”バージョンのアウトラインは、ユーザー部門、およびシステム導入によって影響を受ける他の部門に対して、明快に内容を伝えようとする姿勢が、かなり明らかになっている。

図表 7.6a　<システム導入プランのアウトライン>

旧バージョン：従来型アウトライン

1.　設置環境準備
　1.1　電気系統準備
　1.2　自然環境準備
2.　組立て
　2.1　取付け
　2.2　インターフェイス
3.　コミュニケーション
　3.1　コミュニケーション・プロトコル
　3.2　その他のハード構成
4.　テスト
　4.1　コミュニケーション・テスト
　4.2　メカニカル・テスト
　4.3　ソフトウェアによるテスト

新バージョン：構造化アウトライン

1.　必要な電気系統を取り付ける
2.　適温と清浄さを確保する
3.　カバーとペーパーフィーダーを取り付ける
4.　ケーブルとコネクターを選択し，取り付ける
5.　プロッターをコンピュータに連結する
6.　コミュニケーションおよびプロトコル・スイッチをセットする
7.　コミュニケーション・チェック・プログラムを作動させる
8.　スタート時の問題を分析する
9.　コミュニケーションの問題を解決する
10.　機械的な問題を解決する
11.　グラフィックソフト使用の際のスイッチをセットする
12.　手持ちのソフトでシステムをテストする

図表7.6bは、一般的なビジネスパッケージ・ソフトのための典型的なユーザー・ガイドのアウトラインである。ただし、訂正後のアウトラインには、2つのバージョンがあることに注意して欲しい。1つは経理部門の責任者向けで、専門用語、適用範囲、順序などが工夫されている。もう1つは経理係向けで、システムのオペレーションを中心に構成されている。

従来型のアウトラインは、ライターがマニュアルの対象読者とその役割について何か知ろうとしても、ほとんど役に立たないということに注目していただきたい。

図表 7.6b　<ビジネスパッケージのアウトライン>

旧バージョン：従来型アウトライン

1. 概要
2. 財務特記事項
   2.1 勘定体系
   2.2 勘定科目の説明
3. システムの内容
   3.1 支払い管理
   3.2 予算作成
   3.3 予算管理
   3.4 財務報告
4. 付録：出力サンプル

新バージョン：構造化アウトライン

〈管理職向け〉

1. 財務情報システムの活用法
2. 法律に合うよう勘定科目を定義する
3. 財務計画、財務分析に利用できるよう勘定科目を定義する
4. 財務報告書の設計
5. 現在の支出構造の分析
6. 予算案のシミュレーション
7. 予算案の実施

〈実務者向け〉

1. データ入力
   1.1 売掛金の入力
   1.2 入金の入力
   1.3 買掛金の入力
   1.4 支払い済み金額の入力
   1.5 予算の入力
   1.6 入力の訂正
2. 報告書作成
   2.1 月間財務報告書の作成
   2.2 四半期財務報告書の作成
   2.3 年度末財務報告書の作成
   2.4 損益計算書の作成
   2.5 予算実績対比報告書の作成
3. 付録：エラーメッセージへの対応

7.7　変換：ヘッドラインからアウトラインへ

モジュール化の方法を用いてマニュアル作成をしようとする者は、いきなりヘッドラインを構造化するところからは始めない。ヘッドラインの一覧を作成する一般的な方法は、むしろ、従来型のアウトラインを作成するところから始まる。次に、それを一連のヘッドラインに“分解”していく。その際、各ヘッドラインは、モジュール1つ1つの価値を表現するようにする。

従来型のアウトラインに手を加える場合、たいてい次の2つの変更が同時に行なわれる。

1.ヘディングの表現をヘッドラインのスタイル（文体）とシンタックス（構文）に変える。

2.その時点で予想し得る最良の“モジュール”－モジュールがどのように定義されていたとしても－の記述内容を引き出すようなヘッドラインを書く。

●アウトラインのスタイルを考える

この2つの作業のうち、前者は比較的簡単である。皮肉なことに、報告書やマニュアルを作成する際、従来型のヘディングを書く前に、ヘッドラインを考えてしまう人がほとんどである。たとえばライターは、自分の説明したいことが“計算結果の中の誤りを訂正する簡単な方法”である、ということを心得ているとしよう。しかしアウトラインは“計算エラーのもう1つの訂正手順”となってしまった。また別のライターは“システムＢによって、データ転送の遅れを解消する5つの方法”ということをいいたかったのが、アウトラインでは“システムＢ：データ転送プログラム”としてしまった。つまり場合によっては、ヘッドラインを書くことは、何が書かれるべきだったというもともとのコンセプト作りの段階に一度立ち返って考えることにほかならない。

●モジュールの中身を見透す

2つの作業の後者の方は難しい。ほとんどの人、特にプロのテクニカルライターは、画一的な区切り方でアウトラインを作成することに慣れていない。彼らは、ヘッドラインを使うと、いいたいことがうまく伝わる、ということを知っているのだろうが－事実、多くの優秀なライターは、すでにアウトラインを利用している。ただし、どんなに高度なことをやっているのか、ということに気付かずに使っているライターもいるが－次のような質問に答えるのは、あまり快く思っていない。

“誤りを訂正する簡単な方法”という内容は、1つのモジュールでまかなえるだろうか？　特にある1つのモジュールを“誤りを訂正する簡単な方法－Part2”と呼ぶことは、技術的につじつまが合わないのではないかという話が持ち上がっているときにはどうか。

●独立アウトラインは必要か

1つの結論として、ある企業やライターは、前述した“独立アウトライン”という中間的な段階を設けている。この場合、作ろうとしているマニュアルの再検討を、うまくサポートできるような、明快で、中身の見透せるヘディングであれば、何も同一量の記述内容を引き出すようなものである必要はない。しかし多くの場合、しばらくすると彼らは、この中間的なステップが不必要なものであり、アウトラインに加えられる2つの変更を同時に行なうことは、比較的簡単に実現できることに気付く。そして構造化アウトラインは、モジュールの数に関する予想が後々はずれることになるとしても、なお独立アウトラインよりも格段におもしろく、また便利なものであるということにも気付くのである。

ここで覚えておいて欲しいのは、マニュアル作成のこの段階においては、各ヘッドラインが、

正確に1つのモジュールに対応している必要はまったくないということである。いずれマニュアル作成者は、モジュールと“1対1に対応する”ヘッドラインの書き方を学ぶ。しかし「読み」の正確さについて、あまり神経質になる必要はない。マニュアル作成の次のステップでは、作成された各ヘッドラインが、いくつのモジュールに値するかが明らかになるはずである。

図表 7.7　＜従来型アウトラインの変換＞

従来型アウトライン	構造化アウトライン
Ｉ．導入	
Ａ．背景	1．販売取引をバッチ処理することの問題点
Ｂ．開発秘話	2．コンピュータ処理の３段階の変換 （取引を３つの処理に分解する）
II．操作上の重要項目	
Ａ．一貫性	3．新しいシステムによっていかに操作が簡単になるか
	4．バッチ処理の廃止
Ｂ．データ処理の基本	5．１つの処理をすることで「すべて」のファイルが更新される
	6．データ入力はただ一度ですむ
Ｃ．機密保持	7．アクセスは厳しくコントロールされる
III．機能	
Ａ．販売	8．売上げを記録する
	9．売上げに関連する勘定科目が設定される
Ｂ．財務	10．売上げデータはどのように財務ファイルに渡されるか
Ｃ．在庫管理	11．売上げデータはどのように在庫管理システムに結び付くか

図表7.7は、従来型のヘディングを、ヘッドラインに書き直したものの一部である。各ヘッドラインが1つのモジュールに値しているかどうかは、誰にも分からない。しかしこの時点では、誰もはっきり分かる必要はない。

書き直されたヘッドラインの順番は、もとのヘディングの順番に対応していることに注目して欲しい。ヘッドラインは、同じ順番に並んでいるため、ヘディングのサブのような格好になっている。場合によっては、もとのアウトラインの各部分を置き換える必要がある。設計者がマニュアルから逆戻りやGOTOをなくそうと思う場合、同時に従来のアウトラインの順番を変えなければならないこともある。たとえば、マニュアルを（システム開発の）背景説明から始めなければならない理由はあまりない。たいてい、ソフトのもたらす利益や優位点などから始めるのが適当であろう。

7.8　モジュールの数は予想できるか

経験の浅いライターは、初期の段階においてモジュールのサイズを概算したり、モジュールとして定義されるものが決められたスペース内にぴったり収まる、ということが信じられないようだ。経験豊かなプロのテクニカルライター（彼らはアウトラインをもとにモジュールの長さについての見積りを出してはいるのだが）でさえ、モジュールの長さを正確に予想することは避けて通ろうとする。実際には、各モジュールが決められたサイズに収まるかについて、したがってモジュールの個数についての見積りを出せるようになるまでには、1ヶ月とかからない。

●モジュールのサイズは「上限」にすぎない

ある1つのアウトラインを見ただけで、そのアウトラインをもとに書くべき項目の1つ1つの「正確な長さ」を割り出すことは、明らかに不可能である。しかし幸いなことに、構造化アウトラインを考える段階では、このような見積りは要求されない。

この段階では、モジュールは、サイズの「上限」（例：1ページ、見開き1ページなど）が定められているだけで、画一的なサイズや長さが定められているわけではない。すべてのモジュールを同じ長さにするよりは、長すぎるモジュールをまったく作らない方がはるかに簡単である。

また各モジュールが見開き1ページ（2ページ分）の枠内に収まるものであっても、その長さと内容には著しい違いがある、ということも忘れてはならない。1つのモジュールは、図表のサイズや数によっても違ってくるが、レイアウト・デザインや印刷・製本などの調整によって、少ないときで200ワード（日本語で約500字）、多いときで1200ワード（同じく3000字）くらいになるだろう。

●モジュールの見積りは変更できる

さらに、構造化アウトラインは、各モジュールのサイズを見積る最後のチャンスではない。アウトラインが完成した後に、設計者は各モジュールの簡単なスペックを書くことになるが、この時点で1つだと考えていたモジュールが、実は2つ以上であったと考えざるを得ない場合もある。またもっと後でも、スペックが全部モデルやストーリーボードの形で構成されたときに、見積りを修正するチャンスがもう一度やってくる。

通常、ライターが記述内容をモジュールのサイズにまとめる判断力を養うには、1～2週間もあれば充分である。素人のライターは、ときにプロよりも早くこの技術を身に付けてしまう。というのも、古い習慣を捨てるのに数日とかからないからである。ある概念や手順を説明するのに必要なサイズに対して物理的な限界を設けるという考えは、最初は過酷な制約のように思えるが、すぐに知的な創作活動を促す実用的な規律であることが、おのずと分かってくるだろう。

●余白は多い方がよい

モジュール化されたマニュアルを成功させるために、マニュアル作成に対する方針を1つ変えなければならない。モジュールが小さかった場合に発生するマニュアルの余白を、進んで導入するということである。マニュアル作成の責任者の中には、埋められていない白いページをうらめしく思う者がいる。彼らは、費用の無駄使いばかりを気にして、読みやすさやメンテナンスのしやすさには、なかなか目を向けない。いわゆる“経済的”な印刷方法と呼ばれるものより、この読みやすさやメンテナンスのしやすさが、ときには100倍も1000倍も費用の節約になるということが、彼らには分からないのだ。

“それではモジュール化されたマニュアルには、余白が多くなりはしないだろうか？”と聞かれたら、答えは「たぶんそうなるだろう」である。

第8章 設計II：ストーリーボードとモデルを作成する

8.1　マニュアル作成の問題解決にはモデルが役立つ

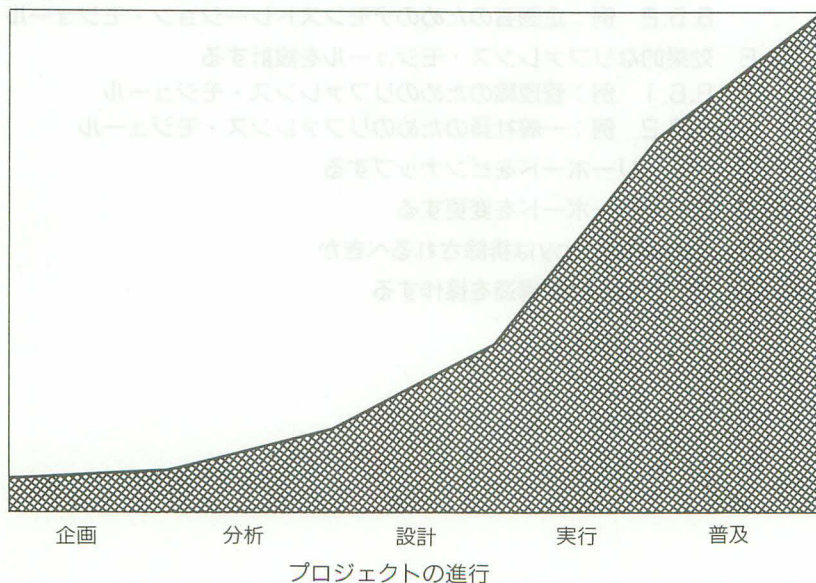
モデルは、お金と労力の節約になる。モデルは、実験的な試みや革新的な試みを、リスクなしで実現してくれる。モデルは、誤りを訂正するのに役立ってくれる。その訂正にかかる費用は、新しいアイデアや製品を試したり、製品の出荷バージョンを修正するのにかかる費用のほんの一部ですむ。モデルなしでは、マニュアルを充分にテストすることができず、欠陥を訂正することさえおぼつかなくなる。

●誤りの訂正は後になるほど面倒になる

図表8.1は、作業全般に関するもっとも重要な関数の1つを表わしている。すなわち、誤りを訂正するためのコストと、その訂正がなされる時期との関係である。この表は、1つの指数として見ることができる。つまり、この曲線は、単に上昇しているのではなく、加速度的に上昇率を増しているのである。

プロジェクトが複雑であったり、またプロジェクトで用いるテクノロジーが、これまで使ったことがないようなもので、しかもリスクをともなうようなものになればなるほど、この曲線の加速度はさらに増加する。それゆえ、非常に単純で手慣れたプロジェクトには、あまり多くの分析と設計は必要ない。ただし、昔からの習慣を再検討することで、もっとも日常的な作業が、根本的に改善される場合もある。

●モデルとは何か

「モデル」とは、あるものを他のものによって表現することである。モデルは、違う材質で（金属製品を粘土で、プラスチックのスイッチを紙で）、あるいは異なるサイズ（ビルの縮小模型、原子の拡大模型など）で作られる。この材質とサイズの違いが、モデルを作りやすく、そして何よりも変更しやすくしている。

図表 8.1　＜訂正にかかるコストの増加率＞

●**マニュアルは1つのモデルである**

モデルは、マニュアルのライターにとって、以下の2点において重要である。まず第一に、マニュアルとオペレーション・ガイド（操作案内）は、それ自体がモデルになり得るということである。もっとも優れた開発グループにおいては、設計チームが“オペレーター・インターフェイス（オペレーターがシステムにどう接するか）”に関する仕様を決めたり、それをテストしたりする1つの方法として、オペレーション・ガイドを書く。言い換えれば、開発チームは、親切なマニュアルを事前に書くことによって、システムが人間と接する部分を定義するわけである。これが結局、システムあるいはプログラムのその後の設計を導き出すのである。

（製品開発の初期の時点でマニュアルが書かれることは、極めて例外的なことである。しかし、もう気付いている会社が徐々に増えているように、マニュアルはなるべく早い時点で書かれるべきである。この方法の数ある利点の中でも特筆すべきは、早い時期にマニュアルを書くことが、とかく後回しにされがちなこと、すなわち「ユーザー」がそのシステムで何をしようとしているかについて注意を促してくれるという点である。）

●**マニュアルにはモデルが必要である**

第二の点は、マニュアルそのものの開発に関係している。つまり、マニュアルそのもののモデルを作る必要があるということだ。マニュアルが数ページ以上になる場合には、草稿を書く前にマニュアルのモデルを作成すべきである。モデルは、各モジュールの内部で何が起こるのかを明確にしてくれるとともに、モジュール間の連結、および結合をすべて明示してくれる。実際、モデルを作成すれば、各モジュールの技術的な内容が正確かどうか評価でき、モジュール間の円環や枝分かれの頻度が予測できるはずである。ソフトウェアの開発と同様、**マニュアルを読み進む上での筋道が多くなればなるほど、マニュアルの信頼性は下がり、読み間違いも多くなる。**

●**モデルはマニュアルをテストする**

モデルは、マニュアルをテストするためのものである。そして、テストの目的は、欠陥や間違いやバグを見つけ出すことである。モデルを作り、それをテストすることは、われわれの誤解を鮮明に浮かび上がらせるものである。また、われわれの意見の相違に焦点をあてるものであり、スケジュールやコストの問題を際立たせるものであり、われわれが欲しているものを手に入れることができないということを、あるいはわれわれが欲しているものを必要なときに手に入れることができないということを証明するものである。簡単にいえば、**モデルは、われわれに誤りを気付かせ、作業のやり直しを促すものである。**

ところが、ライターをはじめDPやMIS*の関係者の中で、モデルの使用を望む者はほとんどいない。私の出会ったクライアントのほとんどが、自分の仕事のどこが悪いのかを知りたがらない。なぜなら、彼らは批評され、試され、検査され、立証され、確認され、評価され、あるいは“なおざり”にされることを嫌うからである。

確かに、批評を好む者はいないということは理解できる。しかし、マニュアルあるいはシステムに長く携わってきた者ほど、批判的な意見を受け入れたがらないのである。だからこそモデルを使って仕事をする場合のもう1つの利点が注目されるのである。つまりモデルは、人が自分のプランに惚れ込んでしまう前に、そのプランの欠陥に気付かせることができるのである。

訳注＊ MIS：management information system。経営情報システム。

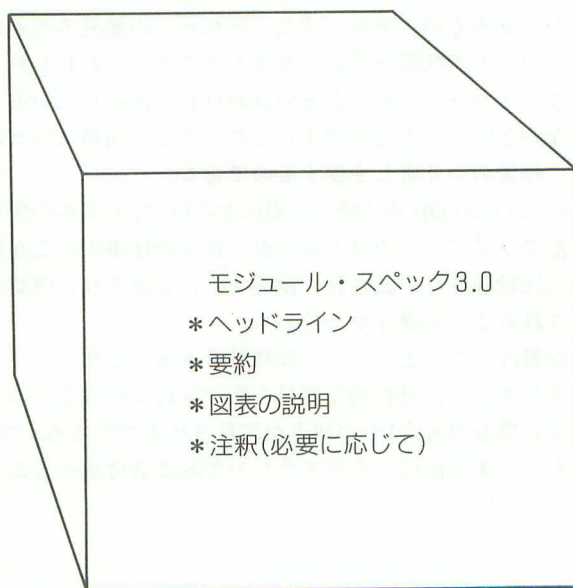
8.2　モジュールごとにモデルを構築する

モジュール・スペックとは、ある1つのモジュールの詳細なプランで、プロジェクトチーム内の技術面の専門家、およびコミュニケーションの専門家によって、共同作成されるものである。各スペックには、ヘッドライン(アウトラインから書き移されたもの)、モジュールの内容を数行でまとめた要約、表や図に関する指示、そして必要ならライターのための若干の注釈、などが盛り込まれる。

●モジュールのスペックを書く

「構造化アウトライン（ヘッドラインのリスト）」は、目的のマニュアルの設計という点で、従来型のアウトラインよりもはるかに明快である。しかし、これはまだ予備的な設計にすぎない。このように、アウトラインだけが準備されている段階では、ヘッドラインに修正する余地はないか、案として出されているモジュールは正しい順番に構成されているか、必要な要素は盛り込まれているか、あるいは不必要な要素は排除されているか、などを確かめることはできない。また、各モジュールに盛り込まれるべき要素が、許されたスペース内に収まるかどうかを、実際に知ることは不可能である。

構造化アウトライン

1.　コンピュータの立ち上げ方
2.　添付ディスクのコピーの仕方
2.1　ディスク・ドライブが1台の場合
2.2　ディスク・ドライブが2台の場合
3.　ファイル名の付け方

図表 8.2　<モジュール・スペックを書く>

マニュアルの真のモデル、つまり、草稿が書かれる前に点検とテストができるような詳細なモデルを構築するには、モジュールごとにプランを策定することが必要である。図表8.2が示すように、各ヘッドラインにはそれ専用のプランである「モジュール・スペック」が与えられる。

スペック（仕様書）を書く紙（あるいは書式）は、モジュールのサイズや内容によって異なる。しかし、見開き1ページの体裁を取るためのもっとも実用的な（しかも手に入れやすい）用紙は、11インチ×15インチのコンピュータ用紙（印刷済み用紙の裏面も可）である。11インチ×15インチシートは、スペックの対象となる11インチ×17インチモジュール（通常のコンピュータ・マニュアルの見開き1ページ）よりもほんの少し小さい。

モジュール・スペックの作成では、次のような作業が行なわれる。

1. **ヘッドラインをモジュール・スペックに書き移す。**
2. **モジュールの内容を1～4行くらいにまとめた要約部分を書く。**
   この要約は、完成されたマニュアルの中で、実際に使用されるものである。これは、それ自体で充分に内容を説明し得る意味がある文章である。したがって“このセクションでは…”とか“下記の手順で…”など、他の文章に注意を促すような要素は含んでいない。むしろ、モジュール全体の要約版とみなされるべきである。科学論文で使われる専門用語でいえば、これは報知的抄録であって、指示的抄録ではない。しかし、この要約の中で、モジュール内の図や表に注意を促す場合もある。
3. **図表を説明する。**
   スペックには、プランの検閲者が見て、図表の出来上がりがどうなるか理解できるように、過不足のない説明が付け加えられなければならない。説明の代わりに、ラフスケッチ、大まかなグラフや表、あるいは図表の内容を明らかにする2、3のキー・フレーズや変数（パラメータ）であってもよい。理想的には、誰もが設計者の望む図表を細部に至るまできちんと仕上げることができるように、充分な説明がなされるべきである。
4. **必要に応じて注釈を付け加える。**
   ヘッドライン、要約、図表の説明だけでは分かりにくい場合には、モジュールに何を盛り込むべきかについて説明する注釈を付け加えることができる。これも、スペックをチェックした者が、完成したモジュールを見て驚かないよう、充分に説明されるべきである。

ほとんどの人は、短時間の“トレーニング”（2～3時間）で、10分～15分もあれば、モジュール・スペックが書けるようになる。同じようなパターンでモジュールが構成されているマニュアルでは、モジュール・スペックの作成は繰返し作業になるので、モジュール1件あたりの作成時間は5分くらいになる。

8.2　モジュールごとにモデルを構築する
8.2.1　どのモジュールにも図表が必要か

おそらく、マニュアル内のどのモジュールにも、図表が付いていた方が好都合だろう。ここでいう図表とは、ダイヤグラムや画面のハードコピー、絵やイラスト、表(言葉や数字による)などである。モジュールの設計がよい場合、図表はあくまで本文の内容を「繰り返して」説明するものであり、書かれていないことを補足したり、追加したりするものではない。

●情報伝達には「redundancy」が必要である

どんなモジュールに対しても、図表が付きものだと考えて欲しい。つまり、各モジュールごとに、最低1つの図表が作成されるよう立案するのである。ただし、どう考えても図表が1つも思い浮かばない場合、また、図表を挿入するスペースが充分にない場合には、モジュール1つに対して図表1つ、という考えを捨てる覚悟を持って欲しい。

図表に示される内容は、本文の内容と部分的に重複する場合もあれば、完全に重複する場合もある。「事実、もっとも完成度の高いモジュールでは、同じ内容の繰返しが行なわれている。つまり、ヘッドラインと要約部分がモジュールの内容を表わし、その内容が図表に反映され、図表は逆に本文の細かい記述に反映され、それによって質を高められるのである。」しかし、このようなredundaucy（繰返し）は、「技術的な情報の伝達は簡潔に」という原則を破るものではないだろうか。

答えは「ノー」である。確かにredundaucyは、経済効率の原則を破り、短期的コストを上昇させるものかもしれない。実際、図表をすべて削除すれば、短期的コストもこれにともない削減される。しかし、**効果的な意思伝達を間違いなく行なうには、redundancyが絶対に必要だということを忘れてはならない**。図表と本文で同じ内容を繰り返すことは、技術情報を伝達するもっとも賢明な方法なのである。なぜなら、読者に対して“違う理解”の仕方を提示することになるからである。

●図表のさまざまな種類

大部分の図表はおもに以下のようなカテゴリーに属する。

・フローチャートおよびプロセス・ダイヤグラム

抽象的なシンボルを利用して、事象や物象の具体的な動き、あるいはデータや概念の論理的な動きを表わしたもの。また、作業手順を明確にするダイヤグラムもこれに属する。

・ディスプレイおよび画面の表示

ビデオ・ディスプレイあるいはインプット／アウトプット・デバイスにそのとき映し出されるものを複写、再生、提示したもの。モジュール1つあたり1つ、あるいはそれ以上の画面を表示することは、オンライン・システムを文書化する上でもっともよく用いられる方法となる。

・ドローイングおよび写真

現実にある物（場合によっては人物）を描写するために用いられる。テクニカル・ドローイングは、対象物（人）を簡単に再現できるため、好んで用いられる方法となっている。一方、写真は、対象物（人）の現実感や信憑性を強調したいときに用いられる。

・“ワード”グラフィック

おもに言葉で作られた表で、これには簡単な飾り、罫線枠、矢印などが付けられる。“ワード”グラフィックは、マニュアルや技術文献ではまれだが、企画書、報告書、教材などでは特に

図表 8.2.1　<マニュアルのための図表>

ダイヤグラム	・フローチャート ・ネットワーク図 ・データ・フロー・ダイヤグラム ・階層構造図/HIPO*
ディスプレイ	・画面 ・ワークシート、フォーム ・ウィンドウ、パネル
絵	・イラスト ・写真 ・デザイン画(投影図)
言葉	・(言葉の)表 ・擬似コードまたは"構造化言語" ・"インフォメーション・マップ" ・リスト、プログラム ・プレイスクリプト
数値	・統計表 ・円グラフ、棒グラフ、折れ線グラフ、面グラフ ・方程式、モデル

有用な方法になり得る（あなたが今読んでいるこのモジュールにも、"ワード"グラフィックが含まれている）。

・プレイスクリプト／ダイヤログ

オペレーター、ユーザー、その他の要員を、あたかも1つのゲームに対する参加者のように思わせる技術である。このプレイスクリプトは、もともと手動のシステムや操作手順のために開発された方法で、次の作業に必要なデータを準備するような手順や、相互に影響し合うような操作手順を説明する場合などに、非常に都合がよい。

・数字および統計に関する表

数学的表現、グラフ、統計表など、科学およびエンジニアリング全般に関する図表を指す。

2つ以上のモジュールに対して、まったく同じ図表を用いてもよいだろうか。答えは「イエス」である。多くのテクニカル・エディターおよびマニュアル作成マネジャーは、この答えに難色を示すであろうが、私が強調したいのは、参照すべき図表が見あたらないという重大なミスを犯すよりも、同じ図表を繰り返し用いる方がましであるということだ。

一方、多くのマニュアル作成者が経験的に気付いていることだが、"同じ"図表を数ヶ所で引用したとしても、事実上参照されるのは、画面上の異なるフィールドだったり、表の異なる箇所だったりする。確かに、現在一般的に用いられている方法では、一度図表を作成したら、それをテキストのさまざまな場所から参照することになる。しかしもっとスマートなやり方をするなら、各参照事項ごとに別個の図表を作成するのがよい。さもなくば、"同じ"図表を使うとしても、その場で参照すべき部分を強調するか、浮き立たせるようにするのがよいだろう。

訳注＊ HIPO：Hierarchy plus Input-Process-Output。ハイポ。データ・フロー・ダイヤグラムでは、処理の流れは分かるが、各モジュールの階層構造がつかめない。逆に階層構造図では処理手順がはっきりしない。これら2つの図の欠点を補い、利点をうまく兼ね備えたのがHIPOである。

8.2　モジュールごとにモデルを構築する
8.2.2　1つのモジュールに多くを詰め込みすぎたら

図表の大きさや位置を調整したり、活字のタイプやページの書式を変えることで、たいていの場合、きっちり収まっているモジュールに、もう少し情報を“割り込ませる”ことが可能である。一般的には、ある概念や処理手順が1つのモジュールに収まり切らない場合は、最低3つのモジュールに分ける必要がある。

●モジュールは画一サイズではない

マニュアル作成者が、標準サイズのモジュールで執筆しようとする際、まず最初に問題となることは、1ページないし2ページの制限内に収めるには大きすぎるモジュールをどうするかということである。

しかし、モジュールの標準サイズは「上限」であって、画一サイズではないということを思い出して欲しい。設計者およびライターがモジュールを考える場合、実際には書くべき内容のひと区切りがモジュール1つ分のサイズで「まかなえる」かどうかを考えているのであって、画一サイズの記述を考えているのではない。結果として、ほとんどのモジュールにはホワイトスペース（余白）ができる。その余白はもっぱら、各モジュールを読みやすくするのに役立つだろう。

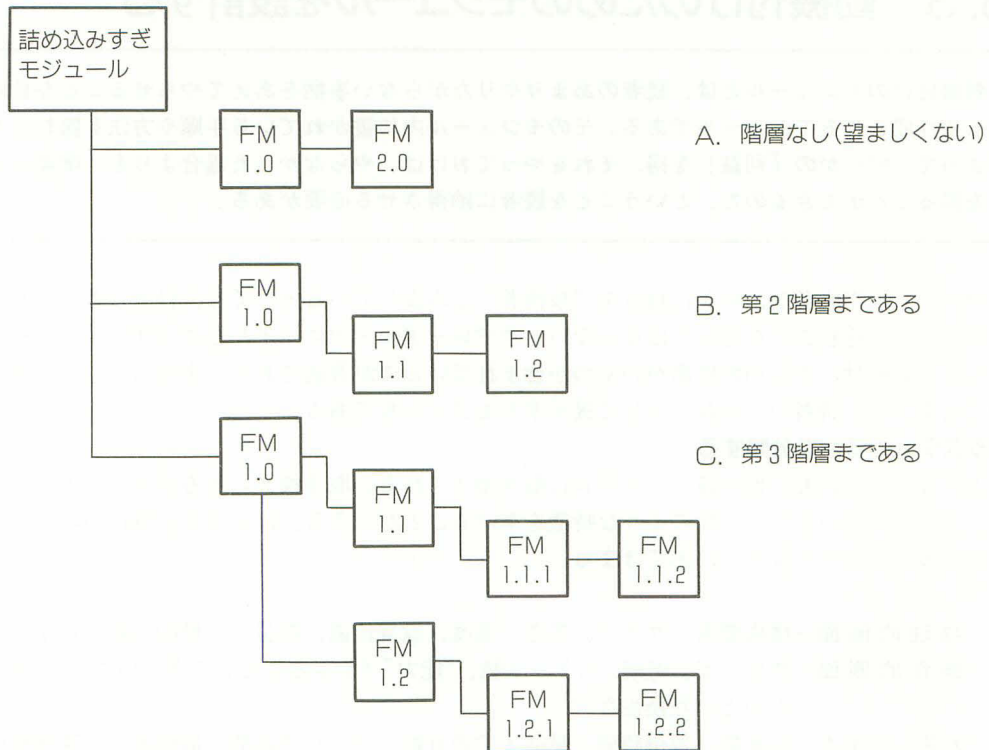
しかしながら、目いっぱいに書き込まれたモジュールについては、改善すべき点が山ほどある。図表、グラフ、イラスト、挿絵などのアートワークを小さくするか、テキストを若干縮小したり、切り詰めたりすることもできる。ただし、ほとんどの出版関係者は、ページによって活字の大きさを変えるのを好まない。

他にも、雑然となったり読みにくくなったりすることなく、モジュールの容量を広げる方法はたくさんある。たとえば、「横に個条書き」にされたようなテキストでは、行数を増やさずに10～20%程度は記述を追加することができる。「均一スペース印刷」にも同じような効果が期待できる。しかし、気を付けなければいけないのは、（アメリカにおける）現在の代表的なオフィス用ワープロは、均一スペーシングを採用して「おらず」、単語間に余分のスペースを取ることで右マージンを調整している、ということだ。こういう場合は、単純に右揃えをやめて、“右側が不ぞろいのまま”の印刷法を採用することで、たいていのワープロでは、1ページあたりの容量を増やすことができるだろう。

詰め込みすぎのモジュールから、内容をいくらか「削る」という方法もある。ただし、削除しても、モジュールの分かりやすさと有効性が失われないことがはっきりしていればの話である。

●大きすぎるモジュールは最低3つに分けられる

ある処理手順や概念が、マニュアルの1つのモジュールに収めるには大きすぎるという事実は、テストケースとして極めて重要なものである。つまりほとんどの場合、その処理手順や概念を1つのまとまりとして見るには大きすぎるということである。特に、モジュールが8.5インチ×11インチのページ2ページ分という大きなものである場合、そのモジュールに収まらないものは、おそらく1つのものではなく、いくつかの小さなまとまりが集まったものとみなした方がよい。

図表8.2.2が示すように、大きな概念を単に2つの段階（フェーズ）に分けても、あまり筋の通った分かりやすいものにはならない。むしろ、まず全体の概論あるいは1つ上のレベルとして、

図表 8.2.2　<"詰め込みすぎモジュール"を分解する>

「その処理手順には2つの段階がある」ということを説明し、次にその2つの段階に1つずつモジュールをあてはめていく方がより分かりやすく、筋が通りやすい。したがって、2つの段階に分かれる処理手順には、3つのモジュール（第一階層1つと第二階層2つ）が必要であり、3つに分かれるものには、4つのモジュールが必要、という具合いになる。

アウトラインの作り方によっては、各ヘッドラインの守備範囲が広すぎて、最初は1つのモジュールに入ると思っていたものを、3段階の階層に分けなければならない場合も出てくる（上図参照）。

確かに、操作手順を1つのモジュールで表わせれば、階層構造や枝分かれを必要とする場合よりも理解しやすく、簡単に実行することができる。そこで、特に詰め込みすぎのモジュールを見つけた場合、マニュアル作成者にとってもっとも賢いやり方は、手順それ自体を変えるよう開発者を説得することである。この方法は、マニュアルの書き方を簡単にするだけではなく、使用されるコンピュータ技術そのものを、もっと使いやすく信頼性のあるものにすることにもなる。

8.3　動機付けのためのモジュールを設計する

動機付けのモジュールとは、読者のあまりやりたがらない事柄をあえてやらせることを目的として作成されるモジュールである。そのモジュール内に書かれている手順や方法を踏むことによって、何らかの「利益」を得、それをやっておけば、やらなかった場合よりも、確実に多くを得ることができるのだ、ということを読者に納得させる必要がある。

マニュアル作成者が、いかに自分を“技術者”とみなしているとしても、自分の考えや方法を読者に売り込むことを怠ってはならない。オペレーター・マニュアルとユーザー・マニュアルには、動機付けのための要素がいくつか含まれているのが普通である。すなわち、システムの「特徴」を、読者の「利益」として提示するモジュールである。

●システムの特性を分類する

どんなシステムも、他の違うシステムに取り換えられる。取り換えられる前のシステムと後のシステムの違いとして、次のような特徴を挙げることができる。ほとんどの特性が以下のそれほど多くないカテゴリーにあてはまる。

・**物理的側面**－構成要素、サイズ、重さ、温度、設置位置、数量、一般的外観、音など
・**操作的側面**－スピード、周期、ステップ数、“能力”（できることとできないこと）、他のものとの互換性など
・**利用しやすさ**－在庫数、習得時間、納品までの日数、サービス期間、取得原価、運転費用など
・**完成度**－簡潔さ、厳密さ、正確さ、精密さ、信頼性、多機能性、拡張性など

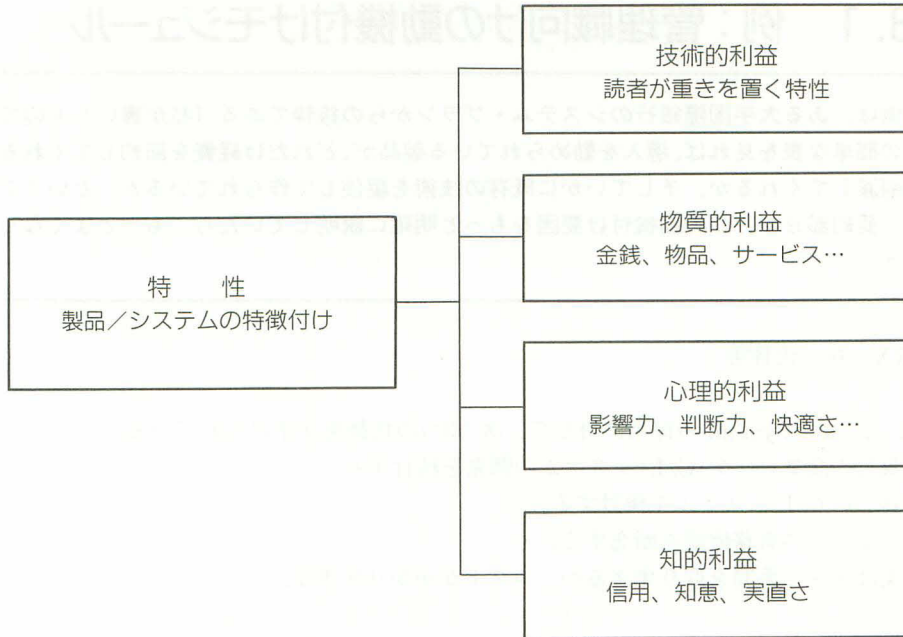
●システムの特性はユーザーの利益

繰り返していうが、システムの交換を行なおうと思うなら、読者に推薦するシステムや手順は、上記の特徴付けのどれかに照らして、取り換えられる前のものと、明らかに異なっていなければならない。そこで問題は、これらの特徴の1つ1つを、さまざまな利益の上に位置付けていくことである。

一番犯しやすいミスは、「ユーザーの能力に関する落とし穴」である。すなわち、多くのライターが、システムにもともと備わっている以上の特性をどんどん積極的に見つけ出してくれる人々がたくさんいる、と考えていることである。しかしながら、そういう人々は、エンジニアやアナリストが考えているほど多くはない。

●どんな利益に訴えるか

もっとも一般的な動機付けは、もちろん「物質的な利益」に訴えることである。書かれている通りにやれば、ユーザーは「お金を儲けるか、節約する」ことができるはずである。一番宣伝しにくいのは、やはり、短期的な高額出費が、その出費を上回る長期的な節約につながる、ということを人々に納得させることである。

物質的利益の他に考えられるのは、「心理的利益」である。すなわち、敬意、地位、名声、知名度、信望、気晴らし、快適さ、影響力、自主性などである。動機付けを専門とする人々によると、これらの心理的利益は、少ない物質的利益よりも－特に準社員（正社員以下）を相手にするときは－重要であるという。たとえば“影響力”は、自分の部署をもっと思い通りにあやつりたいと思っている管理職にとって魅力的である。しかし、同時にそれは、もっと思い通り

図表 8.3　＜動機付けのためのモジュールを設計する＞

に“自由時間”をあやつりたいと思っている事務員にとっても魅力的である。

「道徳心や知性にアピールする」やり方は、、アメリカのビジネス界や政界ではあまり使われない。このアピールとは、正しいから、公平だから、啓発的だから、あるいは気高いから何かをするということである。ある特定の団体（大学や宗教団体など）において、またある文化全体においても、こうしたアピールは、われわれの社会の人間が、損益計算書の最終行に何とかもっとよい結果を書き込みたいとがんばるように、人々を新しい試みへと駆り立てるものである。

強調すべき点は、これらの利益は、システムの特性の中では、何ら具体的には示されないということである。新しいシステムの「コスト面での特性」でさえ、必ずそれが読者に物質的利益をもたらしてくれるという、充分な説明と理由付けが必要になるだろう。

マニュアル作成者は、読者／ユーザーが何を欲しているかを分析し、指示された作業を行なえば、必ず欲しいものが手に入るのだということを－モジュールの要約部分に－明示しなければならない。そして図表は、たいていの場合、その作業を行なったときと行なわないときを並べて、どちらが得かを比較するものでなければならない。

8.3　動機付けのためのモジュールを設計する

8.3.1　例：管理職向けの動機付けモジュール

下の例は、ある大手国際銀行のシステム・プランからの抜粋である（私が書いたものではない）。この簡単な表を見れば、導入を勧められている製品が、どれだけ経費を節約してくれるか、労力を削減してくれるか、そしていかに既存の技術を駆使して作られているか、ということが分かる。要約部分で、この動機付け要因をもっと明確に説明していたら、もっとよくなっていただろう。

3．CRX　対　代替案

CRX　Loan　System の採用に対して、次の3つの代替案が挙げられている。
- 現行の商業ローン（C/L）システムの開発を続行する。
- 新しい C/L システムを検討する。
- いずれかの業務機能を断念する。

代替案はすべて手間を取りすぎるか、コストがかかりすぎる。

CRX　Loan　System は、当銀行の各部署で現在使用しているシステムと同種か、または互換性を持つものである。そのプログラミング言語、レコードのレイアウト、インプット／アウトプットのフォーマットは、すでに他の部署でおなじみなので、CRX を現行のローン手続きと合併させるのは簡単で能率的でもある。下記に示した CRX 導入の利点に注目して欲しい。

- 当銀行のシステム・アナリストは、すでにこの製品を知っている。
- この製品は多くのユーザーによってテスト済みである。
- 出力書式、操作手順、マニュアルはすでに出来上がっている。
- ソフトには、すでに開発料が支払われている。

代替案

現行の C/L システムの開発を続行すると...現行のバージョン2.4を大幅に変えなくてはならない。すべての表示画面に新しいフィールドを加え、マスターファイルを拡張しなくてはならない。この変更は大変手間とコストがかかるため、新型ローンシステムは、完成する前に時代遅れになってしまう。

他の C/L システムを検討すると...検討に時間がかかる。新システムを購入しなくてはならない（一方われわれはすでに CRX を所有している）、そしてさらに学習や指導に時間がかかる。[また、準備費用の査定を見ても、CRX が他より勝っている。]

いずれかの機能を断念すると...高利回りの企業貸付を、あまり大量に記帳するわけにはいかなくなる。一方細かい非能率的な貸付業務を処理する必要性は、これからますます増えていくはずである。

CRX	代替案
＊安価	＊高価
＊速やかな開発が可能	＊開発に時間がかかる
＊テストおよび実証済み	＊無からのスタート
＊システムの専門技術はすでにおなじみ	＊???

図表 8.3.1　<CRX　対　代替案>

8.3 動機付けのためのモジュールを設計する

8.3.2 例：エンジニア向けの動機付けモジュール

下の例は、ある軍事施設に勤めるエンジニア向けのユーザーガイドから抜粋したものである(私が書いたものではない)。動機付け要因は、要約の部分にはっきりと示されている。つまり、技術の説明はもとより、「選択権」ーすなわちエンジニアが自由に仕事のやり方を選べることーが示されている。右ページの図表では、機能の差は比較されているが、選択権に関する動機付けについては強調されていない。

1.3 ランダム・アクセス・カーブ・ファイル*はエンジニアにとって都合がよい

これまでわれわれは、どちらかというと柔軟性を欠くカーブ・ファイルの設計から離れられないでいた。現在、ランダム・カーブ・ファイルのおかげで、自動ユニット変換、有意曲線名や有意パラメータ名、そして、より多彩なサポート・ソフトウェアなどを持つことができる。曲線は、現在最大8個の独立パラメータと50個の従属パラメータを持っている。

かつては曲線は、比較的意味のない“曲線番号”によって認識されていて、その中のデータは、曲線のX，Y，Z，またはW変数と呼ばれていた。気の毒なのは、自分の仲間が度によるALFAでCLALFA曲線を作り、自分のプログラムでは、ALFAはラジアンを返すと想定していた技術者である。あなたが“Z”でなく“W”変数の線をプロットしたいといったとき、なぜプログラマーが頭にきたか？　また、あなたのUFTASプリンター出力が、大量の外挿メッセージ内で失われた回数はどうだったか？

現在では曲線は名前で認識される。(そのため、曲線番号のレンジに関して起こりがちな誤りはもうない)。曲線を作っている変数は、ユニットと同様に、名前を持つ。今では、自分のデータをどのユニットに入力したいかをソフトウェアに指示することができ、残りの処理は、新型の自動ユニット変数機能がやってくれる。

プロット・パッケージがあると、どの変数をX軸として選んでもよいし、その他のどの変数を“〜の線”としてプロットしても構わない。自分のデータをどのユニット内にプロットするかを選ぶことができ、プロット上で自分の変数をどう呼ぶかさえ選ぶことができる。

曲線は、以前よりはるかに大きくすることができる。最高8個までの異なる独立変数の関数である曲線を操作できるようになった。曲線は、多くの従属パラメータ（実際にはそれぞれ50個まで）を持つことができるようになった。

プロット・パッケージに加えて、サポート・ソフトウェアは、UFTAS曲線の作成、検査、組合わせ、印刷、再フォーマットの各機能、およびテクトロニクス・デジタイゼーション・プログラムを内蔵し、それらによって完全でパワフルなパッケージが形成されている。

新しいランダム・アクセス・カーブ・ファイルの利点

機能	古いカーブ・ファイル	新しいランダム・カーブ・ファイル
1. カーブの参照方法	数値	有意名
2. パラメータの参照方法	X, Y, Z, W 変数	有意名
3. パラメータユニット	なし	あり
4. 自動ユニット変換	なし	あり
5. プロットの柔軟性	皆無	はるかに柔軟で多彩
6. 独立パラメータの数	最大３コ	最大８コ
7. 従属パラメータの数	１コのみ	最大50コ
8. アーギュメント呼出しのための検索ルーチン	標準	より柔軟な有意名
9. サポート・ソフトウェア	CREATE PLOT PRINT	CREATE PLOT PRINT AUDIT MERGE CONVERSION DIGITIZING
10. 外挿	必須	ON/OFF の切換え可能
11. 効率	悪い	よい

図表 8.3.2

原注＊ランダム・アクセス・カーブ・ファイル：random access curve file。CAD（コンピュータ援用設計）で製図を行なう場合に使用される曲線の形状を定義する特別な方法。

8.4　初心者のためのモジュールを設計する

初心者のためのチュートリアル（指導用）モジュールでは、ある1つの概念あるいは作業が提示される。その概念あるいは作業は、すでに読者が習得していることであるかどうかが確認されなければならない。マニュアル作成者は、そのモジュールの意図するところを、読者がそこでマスターすべきある特定の項目の観点から定義し、読者はそれを確実に会得することを要求される。その際の方法としては、質問に答えていく、簡単な操作を実行していく、あるいはチュートリアル(プログラミングされた教科書の一種)の指示に沿って進んでいく、などがある。

●読者に問いかけ、実際にやらせてみる

初心者のためのチュートリアル・モジュールには、新しいことが何か1つ含まれている。

ライターが、効果的なチュートリアル・モジュールを設計できるようになるには、そのモジュールから読者に何を学び取らせたいのかを、まず明確にすることができなければならない。そして、この目的をうまく達成しようと思うなら、学び取らせたい概念や考え方に焦点をあてた作業やテストを考えるのが一番便利な方法だろう。読者がある質問に答えられ、ある選択を行なうことができ、ある作業工程を完了させることができ、学ぶべきものをマスターしたと確信できたとしたら、そのモジュールは成功したといえるだろう。もっと高度な教育的記述内容においては、ある作業にどれくらい時間をかけられるか、あるいは答えの中に間違いがいくつまで許されるかなど、いろいろな限定条件が設定される場合もある。

●初心者のためのモジュール

図表8.4aに見る例は、経験のあるオペレーターにとっては、わざわざ質問されるようなことではないかもしれない。しかし、初心者を対象とするときは、これがまさに正しい伝え方なのである。(注：チュートリアル・モジュールは、ごく小さいスペースで足りることが多い。1ページのモジュールだったり、従来の8.5インチ×11インチよりも小さいページですんでしまうこともまれではない。)

●オペレーターのためのモジュール

図表8.4bの例は、オペレーターに関するもっと典型的な記述である。通常、この作業は“解答画面”を作成することであるため、マニュアルは実際の端末操作と並行して使われなくてはならない。(実際にシステムを運用している読者を想定しない限り、一連のチュートリアル・モジュールをどのように提示すべきかを想像することは難しい。特に経験のない読者を相手にする場合、実際に行なわせてみなければ、基本的な作業を習得させることは不可能に近い。)

＜L＞を押すと、ログ・ディスク・ドライブが変更されます。

“L”は何の略でしょう？

____

図表 8.4a

カーソル・キーとインサート・キーを使って、
画面1を画面2に変えてください。

ログ・オン

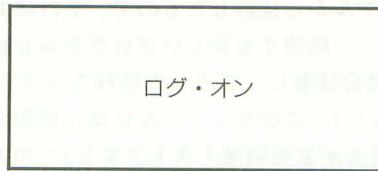
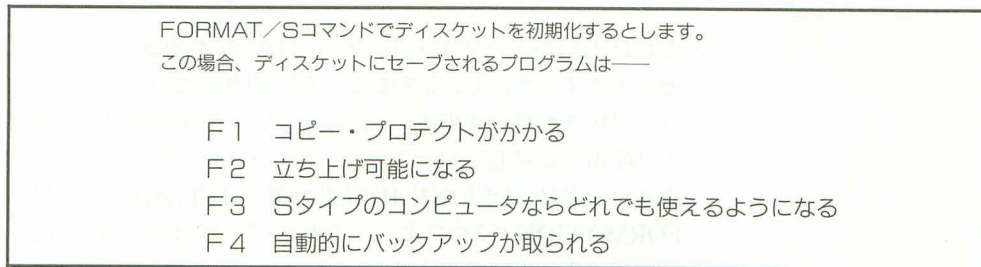
画面1

最初の
ログ・オン

画面2

図表 8.4b

FORMAT／Sコマンドでディスケットを初期化するとします。
この場合、ディスケットにセーブされるプログラムは——

F1　コピー・プロテクトがかかる
F2　立ち上げ可能になる
F3　Sタイプのコンピュータならどれでも使えるようになる
F4　自動的にバックアップが取られる

図表 8.4c

●初心者に効果的なチュートリアル

図表8.4cのような選択肢による簡単な質問形式が、本の中（プログラムに組み込まれた教科書である場合もある）やコンピュータによる一連の操作説明用画面の一部として現われることもある。教科書をプログラムに組み込むことは、初心者を教育するのにもっとも効果的な方法ではあるが、おもしろいことにプログラミングされたテキスト（一般にチュートリアルという）は、設計がよければよいほど、読者にスキップ、ジャンプ、ブランチ、回り道などを強いることとなる！　問題は経験のない、あるいは目の離せない読者である。つまり、このような読者が、プログラミングされたテキストの中で道に迷った場合、後戻りできないということである。この問題の解決策としては、「オンライン・チュートリアル」がある。これもプログラミングされたテキストの一種だが、読者は枝分かれの状態を“意識しなくてもよい”ようになっている。

もっと込み入った記述内容の場合には、明らかにスペシャリスト、つまり教育技術の専門家の技能が要求される。

8.4　初心者のためのモジュールを設計する
8.4.1　例：アナリストのためのチュートリアル・モジュール

下の例は、ある指導用マニュアルから抜粋したもので、それはあるコンサルティング会社の財務アナリストに、統計表を作り、処理する新しいプログラムを紹介する目的で作られた（私が書いたものではない）。この場合読者は、こうした特殊なシステムについては初心者である（データ処理一般についてではない）。このモジュールには、要約の部分がない。したがって、簡単な質問を1つか2つ付けて、読者が変数領域をストアする3つの方法の違いを理解し得たかどうか確認するとよい。

18．変数定義コマンドをストアする3つの方法

変数定義コマンドをストアする次の3つの方法のうち1つを選んでください。

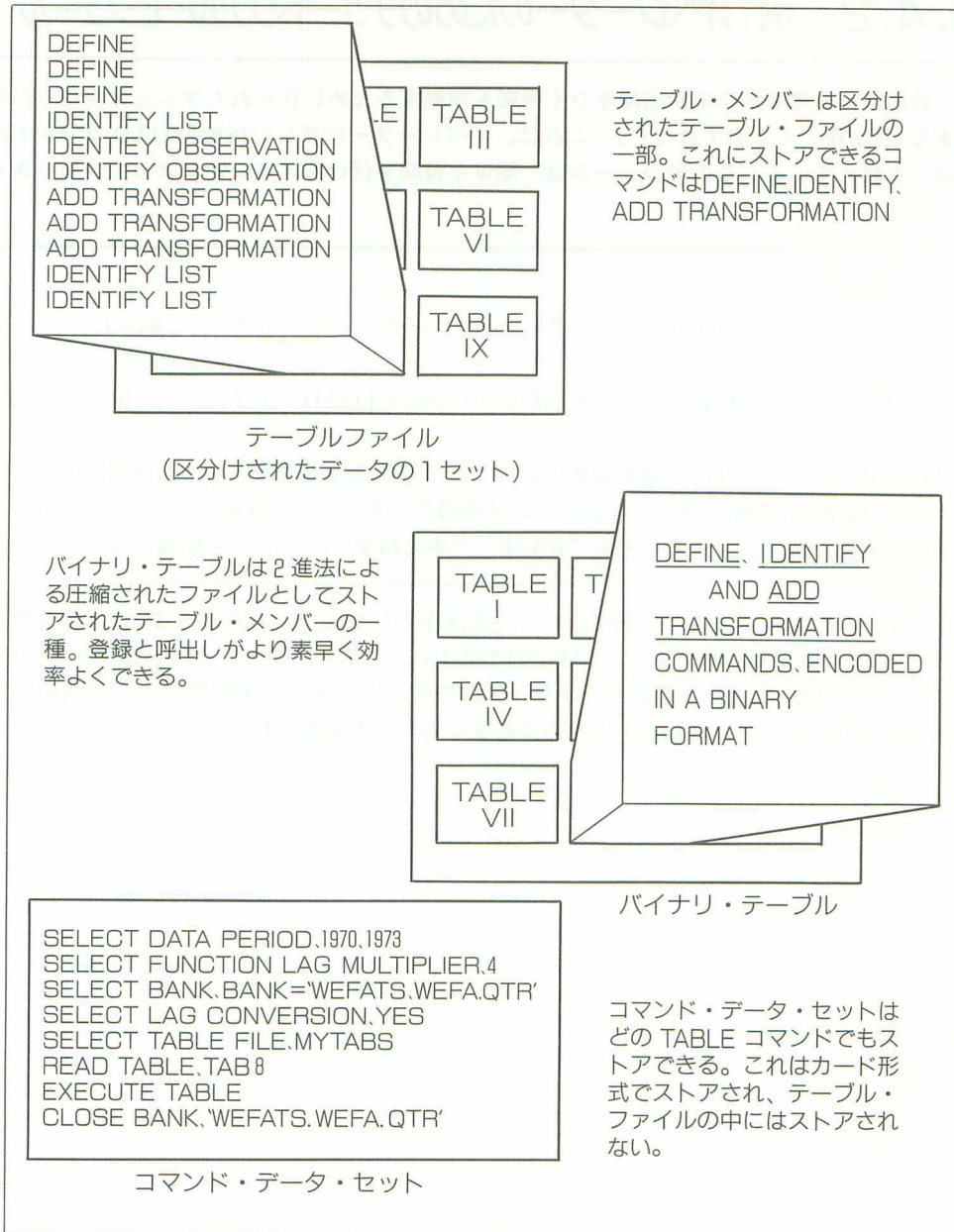
テーブル・メンバー…………これは区分けされたテーブル・ファイルの一部として、ストアされた変数定義コマンドの1セットです。１つのテーブル・メンバーにストアできるコマンドは、DEFINE，IDENTIFY LIST，ADD TRANSFORMATION の3つです。

バイナリ・テーブル…………これは上記のテーブル・メンバーと同じ変数定義コマンドの1セットです。ただし、2進法による圧縮されたフォーマットでコード化されていますので、スピーディーで効率のよい登録および呼出しが可能です。このバイナリ・テーブルにストアできるコマンドは、DEFINE，IDENTIFY LIST，ADD TRANSFORMATION の3つです。バイナリ・テーブルは、変更あるいは編集することはできません。

コマンド・データ・セット…これも変数定義コマンドの1セットですが、カード形式で登録されます。ただし、テーブル・ファイルの一部としては、登録できません。このコマンド・データ・セットには、あらゆるテーブル・コマンドをストアすることができます。コマンド・データ・セットは、編集したり変更したりすることができます。

テーブル・メンバーやコマンド・データ・セットを作るには、UNI-COLL の QED あるいは IBM の EDIT などのテキスト編集プログラムを使います。テーブル・メンバーを登録するためのテーブル・ファイルを作るには、BUILD TABLE FILE を用います。バイナリ・テーブルを作るには、WRITE BINARY TABLE を使用します。

テーブル・メンバーを呼び出すには READ TABLE を用いますが、その前に目的のメンバーをストアしてあるテーブル・ファイルを、SELECT Table File で選びます。バイナリ・テーブルを呼び出すには、READ BINARY TABLE を用います。

〈変数定義コマンドをストアする3つの方法〉

図表 8.4.1

8.4　初心者のためのモジュールを設計する

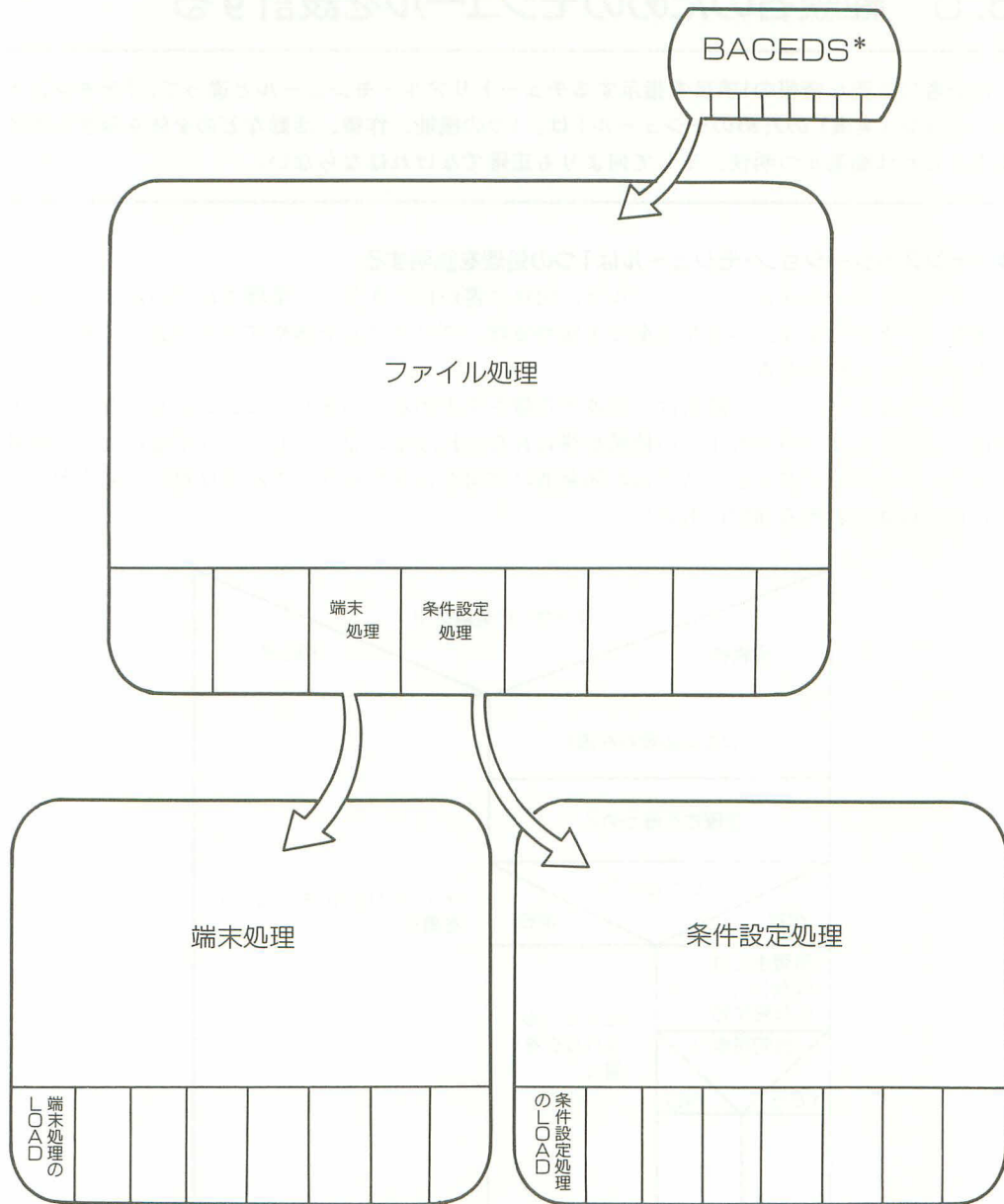
8.4.2　例：オペレーターのためのチュートリアル・モジュール

下の例は、軍事基地に電話回線をひく手順を指導するために作られたマニュアルからの抜粋である（私が書いたものではない）。これは、オペレーターに新しい技術（LOADコマンドの使用法）を教えている。このモジュールは、簡単な質問を付け加えることによって改善できるだろう。

モジュール　11.0　　ユーザー／オペレーター　マニュアル　　84年1月

“Post”から“Base”にデータを動かすために<LOAD>オプションを用いる。

《LOAD》オプションは、《端末処理》からでも《条件設定処理》からでも使用することができる。《LOAD》を呼び出すと、システムは補助データベース“POST”からデータを読み、必要な条件、端末、データベース“BASE”の中に構築すべきものを定義する。

モジュール10、および10Aに説明されている操作をした後、システムの《ファイル処理》の部分にある、《LOAD》オプションを呼び出せばよい。このオプションが呼び出されるたびに、システムは“POST”という補助データベース内のデータを読み、補助データを“BASE”というメイン・データベースに転送するために必要なあらゆる調整を行なう。

図表 8.4.2

原注＊ BACEDS：合衆国陸軍の軍事基地で使用される電話ネットワークを形成するシステム。

8.5　経験者のためのモジュールを設計する

初心者に必要な情報の1項目を指示するチュートリアル・モジュールと違って、「デモンストレーション（実演）のためのモジュール」は、1つの機能、作業、活動などの全体を示すものである。これは簡潔かつ明快、そして何よりも正確でなければならない。

●デモンストレーション・モジュールは1つの処理を説明する

デモンストレーション・モジュールは、明快に書かれ、きちんと整理されていれば、実質的な情報のひとかたまり、つまり完全な手順や処理、プログラム全体やプログラムの各モジュールを表わすことができる。

このようなモジュールの読者は、記述の正確さを求める。つまり、モジュールに書かれた手順通りに行なえば、いつも正しい結果が得られなければならない。もしそうでなければ、読者はマニュアルのライターとシステムの開発者に文句をいうだろう（これと反対に、初心者だったら自分自身を責める傾向にある）。

図表8.5　<デモンストレーション・モジュールを設計する>

図表8.5が示すように、ある1つのデモンストレーション・モジュール、あるいはそれらが複数集まって階層構造をなしたものを企画する場合、まず最初に、対象読者が経験豊かな自信に満ちた者であり、チュートリアル・モジュールの設計のような特別な配慮は、いっさい必要ない者であることを確認しなければならない。

次は、モジュールの設計者の仕事で、説明しようとしているプロセスー通常、人がどのように作業を行なうと推測されるかーの要約を書くことである。この要約は、説明内容や条件付きの処理などの単純明快なリストでなければならない。

驚いたことに、マニュアルやオペレーション・ガイドのライターたちの多くは、説明しようとする事柄の実際の正確なやり方を知らないままに書き始めてしまうことが多い。マニュアル作成者は、これらの要約をただ書けばよいというものではなく、それらをテストしてみる必要がある。

すでに存在している手順を書き上げる場合、テストは簡単である。つまり、われわれは、プログラムあるいはデバイスが期待通りに動くかどうかを見るため、記述の指示通りに（「余計」なことはせずに）やってみてくれる人を探せばよい。システムがまだ開発中の場合は、プロセスの要約は、検閲者によって正確かどうかがテストされなければならない。実地テストで1回だけうまくいったとしても、さまざまな条件下で最高の状態が保証されなければ意味がない。

もちろん、ある特別の場合は、この手順がごく初期のプランあるいはスペックの一部であることもある。この場合は、手順を確認するよりも、開発者がスペック通りに働くシステムを作っているかどうか確認する方が大切である。

●手順が１つのモジュールに収まらなかったら

マニュアル設計時のもっとも興味深い問題は、プロセスあるいは処理が、「1つのレベル」か、あるいは「複数のレベル」の手順かを決めることである。簡単にいえば、モジュール1つに収まるかどうかを決めることである。

もし「1つのレベル」なら、つまり手順に関して言及されるべきすべてのことが1ページあるいは2ページのモジュールで扱える場合には、設計者は要約を書き、手順を簡単に図式化したものをスケッチして、モジュールのスペックを作成する。

しかし、その処理全体が、複数のモジュールを必要とした場合はどうか。

ある1つのプロセスが、1つのデモンストレーション・モジュールには大きすぎるものだったら、それらを階層構造に分ける必要があるだろう。つまり、上部構造のモジュールが1つ必要である。このモジュールは、説明すべきプロセスの1つ1つの構成要素にあてられた複数のモジュールを下部構造として持つことになる（第二レベルまである階層構造となる）。第三ないし第四レベルまでの階層構造になることもある。

プロセスの階層構造あるいは構成要素を定義するには、ちょっとした工夫が必要である。込み入ったものを複数の構成要素に分解するには、つねにさまざまな方法が考えられる。最良の方法は、マニュアルを「使いやすさ」の極みにまで高めること、つまり、読飛ばしや枝分かれや逆戻りの回数をなるべく減らすことである。

8.5　経験者のためのモジュールを設計する

8.5.1　例：管理者のためのデモンストレーション・モジュール

下記の例は、軍需品管理システムのためのオペレーション・マニュアルから抜粋したものである(私が書いたものではない)。この例は、特定のコマンドの使い方を示すものだが、前もって充分なオリエンテーションを行なうことを前提にしている。このモジュールは要約部分を付け加えることで、ガラッと生まれ変わるだろう。

MI
マスター在庫調査

目的：

MIトランザクションは、以下の者が使用するよう設計されている。

・在庫管理担当者
・各基地の倉庫担当者
・各基地の保守用具管理（MMC）担当者

MIトランザクションの使用目的は、以下の通り。

・手持ち在庫品目のチェック
・各品目が保管されている補給機関（倉庫）のチェック
・総合在庫情報チェック

使用するデータベース：

・MSDB－マスターサプライ・データベース

使用可能なトランザクション・モード：

・6（検索用）

必要なトランザクション・キー：

・部品番号・・・・・・・・・・(32文字)
・メーカーコード・・・・・(5文字)
・または、在庫登録番号・・・・・・・(9文字)

エラーメッセージ：

1001－在庫登録番号　入力されたものが見つかりません。
1002－部品番号・製造者　コード入力されたものが見つかりません－チェックにはXPトランザクションを使用してください。
1003－部品番号・製造者コードまたは在庫登録番号を入力してください。
1004－入力した部品番号・メーカーコード用に多重在庫登録番号あり－必要な在庫登録番号を決めるには、XPトランザクションを使用のこと－在庫登録番号でトランザクションを再入力してください。
1005－登録されていない部品番号は入力できません。
1408－補給機関無効－データディクショナリを参照
－補給機関の表示終了
－補給機関リストの次ページを見る場合は、ENTERキーを押してください。

SUPPLY MANAGEMENT
PRIMARY KEY SELECTION SCREEN
RETRIEVE OPTION TRX 6MI KEY
TRANSACTION MASTER INVENTORY INQUIRY
*** ENTER THE FOLLOWING KEY INFORMATION ***
PART NUMBER -------- A8428
MFG CODE ---------- 81535
NIIN --------------- 00 7013496
キー入力
RESET
ALT
ENTER
ENTERを押す
KAF/AD
SUPPLY MANAGEMENT
MASTER INVENTORY INQUIRY
KSXMI01
NSN 5905 00 7013496
PART NUMBER/MFG CODE A8428 81535
NSN DESCRIPTION SWITCH ROTARY
UNIT OF ISSUE EA
NSN INVENTORY CONTROL QUANTITIES
30 ON ORDER
100 MAXIMUM REORDER POINT
50 MINIMUM SAFETY STOCK
SUPPLY ORG | BALANCE ON HAND | ALLOCATED | IN TRANSIT | BACK ORDERED | REPAIR LOCAL | REPAIR ABROAD | CALIBRATION
TOTAL 50 0 10 10 0 0 0
JABER 20 0 5 5 0 0 0
SALEM 30 0 5 5 0 0 0
OPTION TRANSACTION 6MI DATE 08/05/82 TIME 16 10 45 TERMINAL 88010001
KEY 007013496A8428 81535
MESSAGE NO MORE SUPPLY ORGANIZATIONS TO DISPLAY
RESET
ALT
ENTER

MIの画面

図表 8.5.1

8.5　経験者のためのモジュールを設計する

8.5.2　例：企画者のためのデモンストレーション・モジュール

下記の例は、コミュニケーション・プランニング・マニュアルからの抜粋である（私が書いたものではない)。これは、企画者（プランナー）が1つの領域の境界線をどのように定義および再定義するかを示すものである。

5．領域を定義する

5.1　領域の再定義

1つの領域を2つに、2つの領域を1つに、という具合に、既存の領域をいく通りかに定義し直すことができます。また座標や辺も変更することができます。

追加したり、削除したり、変更したりすることで、領域を定義し直すことができます。この場合、システムが以下の情報を入力するよう求めてきます。

入力装置：デジタイザーかキーボードか
更新の種類：追加、変更、削除のどれか
領域更新／座標更新：全座標か一部の座標か
対象領域：再定義の対象となる領域

領域更新

追加：座標を入力できます。作業を終了するときは、デジタイザーから“g”、またはキーボードから“＊”を入力してください。

変更：追加と同じ手順です。新しい座標が古い座標に書き換えられて記録されます。

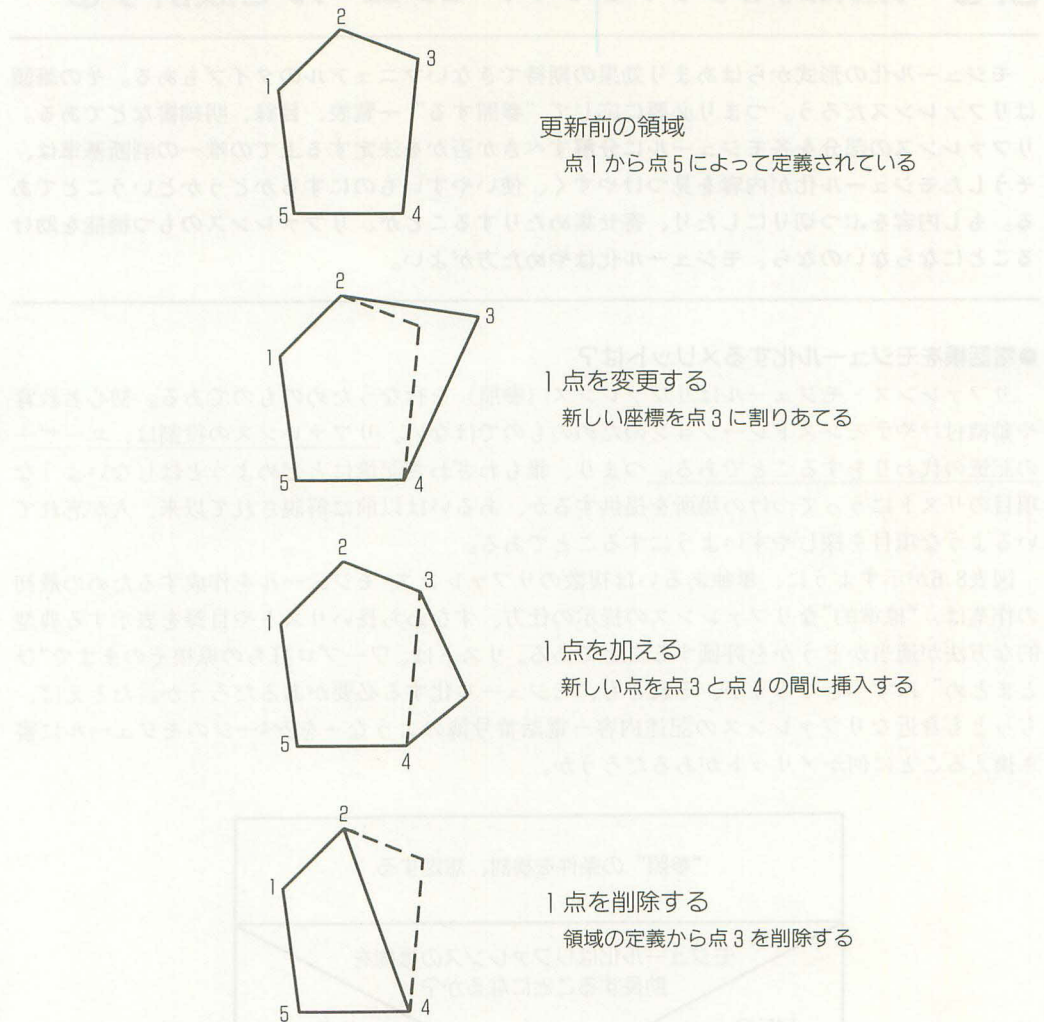
削除：マスターファイルから指定された領域が削除されます。

座標更新

追加：現在の座標が表示されますので、追加したい座標を指定すると、その座標が表示中の座標の後に追加されます。

変更：変更すべき座標を指定し、次に新しい座標を指定します。新しい座標は古い座標に書き換えられて記録されます。

削除：削除すべき座標を指定します。指定を確認すると、その座標は削除され、後の座標がすべて繰り上がります。

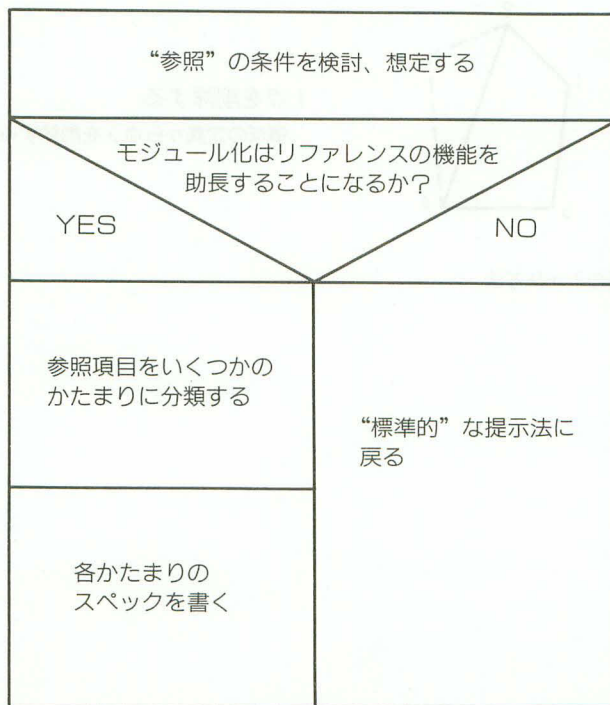
注：領域が再定義されると、システムは必ず影響を受けるすべての構築物、各線分を結ぶ点、端点の割りあてをやり直します。

図表 8.5.2　領域変更の3つの方法

8.6　効果的なリファレンス・モジュールを設計する

モジュール化の形式からはあまり効果の期待できないマニュアルのタイプもある。その筆頭はリファレンスだろう。つまり必要に応じて“参照する”一覧表、目録、明細書などである。リファレンスの部分を各モジュールに分解すべきか否かを決定する上での唯一の判断基準は、そうしたモジュール化が内容を見つけやすく、使いやすいものにするかどうかということである。もし内容をぶつ切りにしたり、寄せ集めたりすることが、リファレンスのもつ機能を助けることにならないのなら、モジュール化はやめた方がよい。

●電話帳をモジュール化するメリットは？

リファレンス・モジュールはリファレンス（参照）を行なうためのものである。初心者教育や動機付けやデモンストレーションのためのものではない。リファレンスの役割は、ユーザーの記憶の代わりをすることである。つまり、誰もわざわざ記憶にとどめようとはしないような項目のリストにうってつけの場所を提供するか、あるいは以前に解説されて以来、人が忘れているような項目を探しやすいようにすることである。

図表8.6が示すように、単独あるいは複数のリファレンス・モジュールを作成するための最初の作業は、“標準的”なリファレンスの提示の仕方、すなわち長いリストや目録を表示する典型的な方法が適当かどうかを評価することである。リストは、ワープロ打ちの原稿そのままで“ひとまとめ”にすべきだろうか、それとも、モジュール化する必要があるだろうか。たとえば、もっとも身近なリファレンスの記述内容－電話番号簿のようなーを2ページのモジュールに書き換えることに何かメリットがあるだろうか。

図表 8.6　<効果的なリファレンス・モジュールを設計する>

多くの場合、何もメリットはない。かつて“論理的にグループ分けされた”リファレンス・リストを見たことがあるが、読者側の使い勝手からすると、逆効果のようだった。たとえば、あるマニュアルでは、エラーメッセージにコード番号が付けられ、オペレーターのミスによって起こったエラーと、システムの動作不良によって起こったエラーとに分けられていた。この場合、残念ながらオペレーターは、画面に現われたものを見て、どちらに分類されるエラーか分からず、結局「両方」の場所を探すことになってしまうだろう。

●リファレンス・モジュールはどこにどう置かれるべきか

出来のよいリファレンス･モジュールは人に何かを教えようとは「しない」。このモジュールの特徴の1つは、ヘッドライン、要約などが非常に短く、また、しばしば「要約のほかにはテキストが何もない」、ということである。リファレンス･モジュールの典型的なものには、完成時において、2ページにわたる図表（チャート、テーブル、リストなど）が含まれることが多い。

リファレンス部分の記述内容が多いマニュアルーモジュール化されているいないにかかわらずーは、あまり効果的なマニュアルとはいえないだろう。リファレンスとは、ユーザーがシステムあるいは製品の動かし方を知った「後」で、必要とされるものである。それまでは、リファレンスの記述内容は、ユーザーにとってあまり親切なものではなく、ユーザーが読むのをためらうものでさえある。

リファレンス的な記述内容を増やすような方針の恐ろしいところは、「用語解説で説明を済ませ」ようとすることである。たとえば、あるマニュアルを、あるユーザー（仮称U）にあてて書いたとしよう。この場合、新しい用語を導入するたびに、用語解説（たぶんマニュアルの前か後ろかにある）を参照することを強要してはならない。用語解説は、チュートリアルやデモンストレーション・モジュールの中で習ったことを「思い出す」場合に役立つものである。読者に用語解説を参照させたり、あるいは、読者が頻繁に用語解説を見ることを想定したりすることは、そのマニュアルが他の誰かのために設計されたものであると読者に思わせることである。

リファレンス・モジュールは、単独では何も教えることはできない。また、何かを教えるための記述の部分に組み込まれるべきでもない。ユーザーにとって、頻繁に使う表を見つけるために、チュートリアル・セクションをいちいち探すのは面倒なことである。

したがって、リファレンス・モジュールは、すぐに探せる場所にあるべきである。マニュアルの最初か最後、あるいはバインダーのカバーの部分でも構わない。ポスターや折込みや“参照しやすい”ようにポケットサイズに折りたためるようなページなどの形を取ることもできる。オペレーターはよく自分でリファレンスを作り、ノートに保管するか、入力装置の脇にテープで貼り付けるかしている。

もしユーザーやオペレーターが、「自分自身」のリファレンス･マニュアルを作っているとしたらーそしてもし、われわれがリファレンス・マニュアルの正確さとメンテナンスのしやすさを望んでいるならー彼らが必要としているものが何かをつきとめ、それを彼らに提供すべきである。

8.6　効果的なリファレンス・モジュールを設計する

8.6.1　例：管理職のためのリファレンス・モジュール

下記は、大規模メーカー向けの電子メール・システムのエンドユーザー・ガイドから抜粋したものである（私が書いたものではない）。これは、全体を概観したり、記憶する助けにはなるが、それ自体は何かを指導したり、実演（デモンストレーション）してみせたりしないことに注意して欲しい。システムの使い方を知っている者だけが、この表を使うことができる。（また、このモジュールは、1ページに収められたことにも注意すること。）

MAIL システムでどんなコマンドが使えるか？

MAIL システムで使用される基本的なコマンドはいくつかありますーユーザーは、送られてきた MAIL メッセージを読んだり、他のユーザーにメッセージを送ったり、古いメッセージを削除したりできます。各コマンドについては、それぞれ別に後述します。

MAIL システムを使うには、読取り、送信などのコマンドをキーインし、<RETURN>キーを押します。コマンドは必ず MAIL>プロンプト時にキーインします。

図表 8.6.1

MAILコマンド一覧

コマンド	意味
BACK	ディレクトリ内の１つ前のメッセージを表示する。
DELETE	一番最後に表示したメッセージを削除する。
DIRECTORY	メッセージ一覧表を作る。
EXIT	MAIL操作をやめる。
FILE	最後に表示されたメッセージを特定ファイルにコピーする。
FORWARD	最後に表示されたメッセージを他のユーザーに送る。
HELP	MAILの使い方についての情報を表示する。
NEXT	現在表示中のメッセージの途中から飛んで、ディレクトリ内の次のメッセージを表示する。
QUIT	実行した削除をキャンセルし、MAIL操作をやめる。この場合、ディレクトリは正確にMAIL操作に入ったときの状態になっている。
READ	ディレクトリ内の次のメッセージまたは現在表示されているメッセージの次の画面を表示する(〈RETURN〉キーを押してもREADと同じ効果)。
READ MAIL	MAILシステム操作中に受け取った新しいメッセージを表示する。
REPLY	最後に表示されたメッセージの送り手に、同じテーマについてのメッセージを送る。
SEARCH	指定された記述内容を含むメッセージを探し、表示する。
SEND	メッセージを他のユーザーに送る。
SEND／LAST	今送ったばかりのメッセージのコピーを別のユーザーに送る。

8.6　効果的なリファレンス・モジュールを設計する
8.6.2　例：一般社員のためのリファレンス・モジュール

以下は、商用ソフトの導入マニュアルからの抜粋である（私が書いたものではない）。これには、同じシステムの使い方をどこかで習ったことのあるワープロ・オペレーターなどに便利なコマンドの機能一覧が付いている。

10．RUNOFF ワードプロセッサー

10.2　コマンドの定義

RUNOFF は、実ファイル項目に登録された情報を処理するものである。この情報は、テキストと RUNOFF コマンドからなっている。

RUNOFF コマンドは、必ずピリオド（.）を先頭に付けて使用する。コマンドを使用した行にはテキストを含めてはならない（すなわち、コマンドがテキスト行に現われることはない）。複数のコマンドを1行に並べることもできる。
RUNOFF は、情報を2つの方法のうちの1つで処理する。“充填”モードでは、その行にそれ以上単語が入らなくなるまで、一度に単語がプリントされる。マージン調整オプションがある場合には、単語間にアットランダムにスペースが追加され、右マージンの調整が行なわれる。このモードは、センテンス形式のテキストによく使われる（今読者が読んでいる文章は、“充填”と“マージン調整”の2つのモードで処理されたものである）。このモードを使えば、行末を気にせず、自由にテキストを入力することができる。“非充填”コードでは、項目単位で1行が処理される。このモードは主として、図表処理用に使われている（このマニュアルの右ページのほとんどがそれである）。

図表 A は RUNOFF コマンドの簡単な一覧表である。図表 B はこのページを構成している項目の割付表である。

BEGIN PAGE(.BP)	BREAK処理し、改ページする。
BREAK(.B)	次行を処理する前に一部だけいっぱいになった行を出力する。
CAPITALIZE SENTENCES(.CS)	各文の最初の単語を大文字にする。
CENTER(.C)	指定されたテキスト行を中央に持ってくる。
CHAIN({DICT}ファイル名　項目コード)	指定されたテキストファイルにつなげる。
CHAPTER title	章の見出しの番号を付け、フォーマットを決める。
CONTENTS	CHAPTERコマンドおよびSECTIONコマンドの実行によって累積した目次を印刷する。
CRT	ユーザーの端末への出力を指定する。
FILL(.F)	はみ出しがないようにして行を埋める。
FOOTING	各ページの最後に指定行を印刷する。
HEADING	各ページの頭に指定行を印刷する。
INDENT N(.In)	指定行を 'n' スペース分頭下げする。
INDENT MARGIN(.IMn)	左マージンの位置決めをする。

INDEX text	インデックス・リストに“テキスト”を登録する。
INPUT	端末から指定行を読む。
JUSTIFY(.J)	右マージンの位置決めをする。
LEFT MARGIN n	左マージンをセットする。
LINE LENGTH n	行の長さをセットする。
LOWER CASE	(指定されたものを除く)すべての文字を小文字で出力する。
LPTR	システム・プリンターに出力命令を出す。
NOCAPITALIZE SENTENCES(.NCS)	CAPITALIZE SENTENCESモードを解除する。
NOFILL	FILLモードを解除する。
NOJUSTIFY(.NJ)	JUSTIFYモードを解除する。
PAGENUMBER n	現在のページ番号をセットする。
PARAGRAPH(.Pn)	(最初の行を頭下げして)テキストをパラグラフにまとめる。
PRINT INDEX	INDEXコマンドで作成され、ソートされた見出し語を印刷する。
PRINT	ユーザーの端末に指定行を表示する。
READ({DICT}ファイル名　項目コード)	指定されたテキスト項目を読み取り、処理する。
SECTION level title	'n' レベル下げて次項の番号を付ける。
SET TABS n{n}...	タブ位置をセットする。
SKIP n(.SK n)	'n' 行分の空白を印刷する。
SPACING n	行間隔をセットする。
STANDARD	デフォルトパラメータを解除する。
UPPER CASE(.UC)	文字をそのまま(大文字または小文字で)印刷する。

図表 A.　RUNOFFコマンド一覧

```
001 .BP
002 .F.J
003 .READ FOOTING.L
004 .INDEX 'RUNOFFコマンドの定義'
005 .SECTION 2 コマンドの定義
006 .SK 2
007 RUNOFFは、実ファイル項目に登録された情報を処理するものである。
008 この情報は、テキストとRUNOFFコマンドからなっている。
009 .SK 2 010
010 RUNOFFコマンドは、必ずピリオド（.）を先頭に付けて使用する。
011 コマンドを使用した行にはテキストを含めてはならない（すなわち、コマ
012 ンドがテキスト行に現われることはない）。
013 複数のコマンドを１行に並べることもできる。
014 .SK
015 RUNOFFは、情報を2つの方法のうちの１つで処理する。
016 “充填”モードでは、その行にそれ以上単語が入らなくなるまで、一度に
017 単語がプリントされる。
018 マージン調整オプションがある場合には、単語間にアットランダムにスペ
019 ースが追加され、右マージンの調整が行なわれる。
020 このモードは、センテンス形式のテキストによく使われる（今読者が読ん
021 でいる文章は、“充填”と“マージン調整”の2つのモードで処理された
022 ものである）。
023 このモードを使えば、行末を気にせず、自由にテキストを入力することが
024 できる。
025 “非充填”コードでは、項目単位で１行が処理される。
026 このモードは主として、図表処理用に使われている（このマニュアルの右
027 ページのほとんどがそれである）。
```

図表 B.　RUNOFF項目割付サンプル

8.7 ストーリーボードをピンナップする

モジュールのスペックは、まだ山積みにされている状態である。このままではテストするにも、手を加えるにも、何の機能も果たさない。そこで次のステップは、このスペックの山を壁に貼り、スペックの“ギャラリー”あるいは“ストーリーボード”を作成することである。このようにしてスペックは、それを書いた“執筆者”、出来上がったマニュアルを読むことになる読者、そして組織内の役員も含めた人々によって見直され、訂正されることになる。

●モジュールのギャラリーを歩く

個々のモジュール・スペックは、一番よい順序で部屋の壁に貼られることによって、“ギャラリー”に作り変えられる。マニュアルのアウトラインを目で見える形にするというこのプロセスは、一般的に“ストーリーボード作業”と呼ばれる。これは映画産業から持ってきた用語である。(おもしろいことに、映画製作向けの技術を用いると、マニュアル作成者にとって、マニュアルというものは、情報の「階層的」集合というより、情報の「連続」と考えられるようになる。)

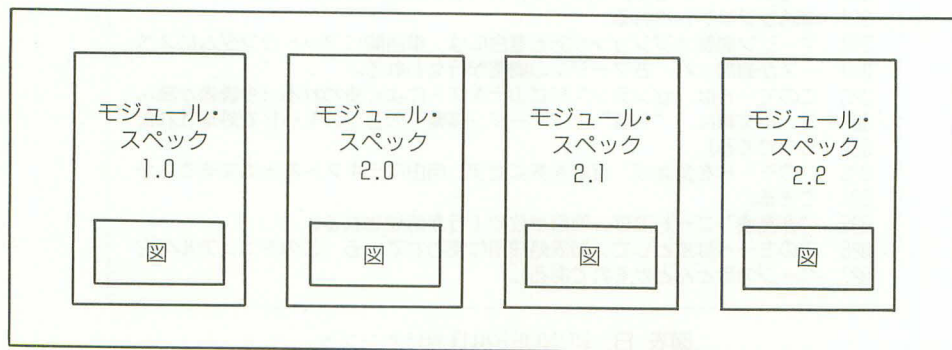
このようにして初めて、スペックを書いた1人あるいは2人は、実際にそれを見ることができる。彼らは互いに質問を交わしながら、ギャラリーを“ウォークスルー*”し、各モジュールの強調すべき点や守備範囲、順序を検討する。

次に“執筆者”－本文や図表の抜けている部分を書き足す人たち－を招待して、ストーリーボードを見直してもらい、一層掘り下げた修正や示唆をしてもらう。

企画者と書き手が納得すれば、次にストーリーボードをユーザーの目から批評してもらう。あらためて、実際のユーザー、あるいはユーザーの事情がよく分かっている人が、ストーリーボードを見直して、明快かどうか、読みやすい並び方かどうか、適切かどうかを調べる。

ユーザーやオペレーター（またはその代弁者）がストーリーボードを見直しているときには、マニュアル企画者は立ち会うべきである。質問がなされれば、企画は明快になり、一貫性が出てくる。また、対象となる読者が実際に何を知っているか、あるいは何を行なうかについての誤解を正すこともできる。マニュアルのストーリーボード・バージョンが、システム開発のごく早い時期に用意されれば、システムの企画の改善方法を見つけ出すことが、事実可能である。

●体の動きに注目せよ

マニュアル設計者は、ユーザーやその他の読者がプランを見直している間、彼らを「観察」

図表 8.7　<モジュール・スペックの“ギャラリー”>

することが大切である。構造化されたプログラミングやその見直し、というような比喩的な意味の探索とは違って、ストーリーボードを使うと、関係者は文字通りマニュアルのモデルを「ウォークスルー」させられることになる。そして、**マニュアルの設計者は、見直す人たちの（目や足などの）身体的な動きをただ見るだけで、設計上の欠陥を見分けることができる。**設計のこの段階でも、オペレーターやユーザーがマニュアルの論理性を追いかけるところを設計者が観察すれば、一貫性のなさ－読み返しや回り道－が明白になってくる。まだ変更が容易なうちに、最悪の設計ミスや、まったく信頼性のない個所が明らかになる。

●１つのモデルは１ヶ所に

ストーリーボードを作成して、それを最大限有効に活用するためには、「1つ」のストーリーボード・モデルを「1つの場所」に貼るのがよい。会社の規模が小さく、どうしても専門の部屋を取るのが難しい場合を別にすれば、ほとんどの会社ではこれは問題ないだろう。しかし、会社の規模がもっと大きくなると、出来上がったマニュアルに関心を持つさまざまな見直し役は、数ヶ所に分かれて作業をすることになる。最大手の企業になると、マニュアル作成部門が、開発者やユーザーから数千マイル離れている、ということもあり得る。

場所の問題については同情するが、それでも「1つの」モデルを「1ヶ所」に、ということを勧める。原則としては、どのような技術出版物に関しても、複数の見直し原稿を作らない、また原稿を郵送しない、ということを勧める。私の体験した技術文書の見直し作業のうち、唯一完全なものは、関係者全員の在席のもとに、多くの質問や討論を繰り返し、必要な人とデータを脇に置いてなされた。

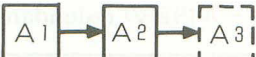
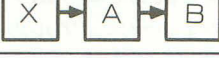
有意義な変更がすべて取り入れられて、最終的に設計者たちが満足すれば、最後にその設計案に署名し、企業の役員（または役員会）を呼んで、ストーリーボードを見直してもらう。設計案が承認されれば、それはそのまま「凍結」する。極端な例外を除いて、変更はいっさい認められない。モジュールの内部での変更ということ以外は、提案された変更が1つ以上のモジュールに影響するようであれば、再びストーリーボードに戻る。

訳注＊ウォークスルー：walkthrough。プログラム開発を効果的に進める技法の１つ。開発プロセスが独善的にならぬよう、プロセス全体に対し、各担当者が、一定の基準やルールに従って何度も見直しする。本書全体がいわばマニュアル作成のウォークスルーを指南したものといっても過言ではないだろう。ただしウォークスルーは設計の妥当性を再検討することであり、修正ではないことに注意。

8.8　ストーリーボードを変更する

構造化設計や映画の構想作りなどにも見られるように、モジュールのストーリーボードを作成することの最大の利点は、それを使うと変更が簡単であるということだ。アウトラインの中に見出される、全範囲に渡る技術的な欠陥やコミュニケーションの方法に関する欠陥が、比較的わずかな“場所の移動”によってはっきりしてしまう、というのは皮肉な話だ。あまり通りたくない悪路ーつまり読者をGOTOに送り出すような個所(回り道、複雑にからみ合った環状の筋道、始まったとたんに終わってしまうようなもの）ーは、モジュールのスペックを操作するだけで、発見し修正することができる。

●ストーリーボードを見直す基準

ストーリーボードは「テスト」であり、テストにはすべて標準形式と評価基準がある、ということを忘れてはならない。マニュアルの循環や迂回の数ーいわゆるGOTOーを抑えなくてはならないと、すでに繰り返し強調してきた。このような大ざっぱな考え方や制約で充分な場合もある。なぜなら、あまり厳密さを要求されない見直し作業では、目標は定められた標準形式に達することではなく、見直す人たちを納得させればすむからである。

しかし厳密なルールにのっとった見直しでは、何らかのはっきりした評価基準が必要である。設計や手順を“最高”のものにしたいときはなおさらである。中心となる基準として、次の定義を提案したい。

「GOTOなきマニュアル」では、読者は読み始めれば、そのまま「読み終わる」はずである。それ以上の情報が必要だったり、読みたいと思ったときは、別のモジュールの初めに移ればよい。そしてそれを読み終えればよい。さらにいえば、たいていの場合、2番目のモジュールは、1番目のが終わったすぐ後に続けられるのが普通である。

要するに、このような制約があれば、マニュアル作成者やテクニカルライターは、読者がモジュールを中途で抜け出したり、モジュールに中途から入ったりすることのないよう、気を配るようになる。特に、あるモジュールの中途から他のモジュールの中途に移らせ、その後、最初のモジュールの出口に戻らせる、というようなことをしなくなる。

さて、先にマニュアルの“分割”について論じてすでに明らかにしたことであるが、ある特定のマニュアルの読者が多様であればあるほど、読者がそのマニュアルをどのように使用するかを予測するのは難しくなる。したがって原則的には、すべての読者を対象として、上に定義したような基準を満たすマニュアルを開発するのは不可能である。そして読者の多様性が著しくなればなるほど、基準の達成は難しくなる。

しかし、出来上がりのマニュアルが、できるだけこの基準に近づくように、モジュール・スペックを作り直し、配列し直すのがマニュアル設計者の仕事である。適切とはいえない個所で読者がモジュールから出たり、モジュールに入ったりせざるを得なくなるたびに、設計の変更を試みるべきだ。(第一稿を書き終わった後では遅すぎる。)

図表 8.8　<ストーリーボードを変更する>

●ストーリーボードの変更テクニック

モジュール・スペックのごくわずかな“移動”だけで、非常に込み入った変更ができるのには驚く。

解　体－1つのモジュールを、列または階層ごとに、2つ以上のモジュールに変換する。それぞれのモジュールに新しいスペックを付ける。
統　合－同じ論点または概念であれば、2つ以上のモジュールを1つにする。
挿　入－ギャップを埋めるために、1つまたはそれ以上のモジュールを追加（挿入）する。
削　除－“論理性”から“読みやすさ”へという趣旨で、モジュールを取り除く。
再配置－1つのモジュールまたはモジュール群を、同一マニュアル内で移動させる。

繰り返していうと、これらの“移動”で、可能な変更のほとんどの説明がつく。（モジュール「内」での変更ということもある。モジュール・スペックの内容をほんの少し変えるか、強調のために注釈を追加する、といった方法だ。）

ストーリーボードは、変更や改訂のプロセスを容易にするために作り出されたテクニックである。ストーリーボードのプランは、1日に何十回も根本的に改訂することができる。しかし、第一稿が完全に出来上がってしまうと、つぎはぎや詰め物で修繕はできるが、根本的に設計をし直して、欠点を取り除くことは絶対にできない。

8.9 Redundancyは排除されるべきか

皮肉なことに、モジュール化されたマニュアルが成功するか否かを確かめる目安の1つは、読者が redundancy に気付く－あるいはときに文句をいう－かどうかである。複数のモジュール間にまたがる redundancy は、枝分かれ、逆戻り、回り道などの必要を緩和してくれる。また、モジュール内部の redundancy は、読みにくさや読み落としなどを補ってくれるものである。

●redundancyとは何か

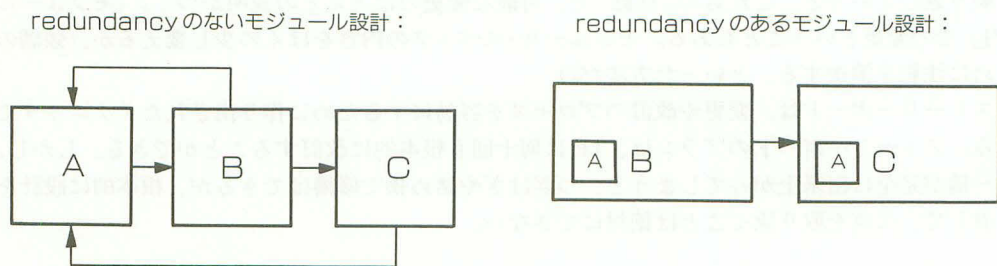
いくぶん呑み込みやすくするために、redundancy 以外の言葉を使うことができたかもしれない：

たとえば、繰返し、敷衍、念押し、言い直しなど。

しかし、redundancy とは何だろう。それは必要以上に用いること、必要以上に消費すること、同じような情報を3回4回5回と書くこと、つまり同じ節、段落、図表を使うことなどである。実際、マニュアルを作成する上で問題が起こった場合、同じ論題を違った言葉や図で説明するよりも、**同じ説明を繰り返す**方がよい。

使いやすいマニュアルの redundancy には2種類ある。モジュール間にまたがるものと、1つのモジュールの中で現われるものの2つである。図表8.9では、モジュール間にまたがる redundancy の単純なものを示している。いろいろな操作を始める前に、しなくてはならない手順というものがある。redundancy のないマニュアルの場合、読者は何度も冒頭の手順に戻らなくてはならない。（タスク B の進め方を習う前に、タスク A のところを読むよう指示され、C、D、E、F・・・についても同様の指示が繰り返される。）それに引き換え、redundancy のあるマニュアルでは、後の操作の個所でも、最初の手順の説明が、決められた表現で繰り返されている。この方法は、たとえば計算機のマニュアルなどでよく使われる。計算機のマニュアルの場合、それぞれの操作を始めるとき、計算機を立ち上げたり、登録を消去する場合の注意書きが付け加えられる。

●プログラムのredundancyとマニュアルのredundancy

マニュアルにredundancyを取り入れるべきか否かの問題は複雑で、議論の的になるところだ。特に興味深いのは、redundancyを取り入れると、モジュール化されたコンピュータ・プログラムとモジュール化されたマニュアルの類似性が破壊される、という議論だ。ある見方によれば、構造化されたプログラムの基本は、プログラムになるべく redundancy が「ない」、というものだ。つまり、ある特定の作業あるいは機能が必要になるたびに、プログラムの中の特定の場所から、それらを呼び出すわけである。

図表 8.9　＜モジュール間の redundancy＞

しかし、もっと綿密に検討すると、類似性はやはり存在する。本当の問題は、ユーザーのために作ったマニュアルが、ユーザーのために実行される“呼出し”をすべて備えているかどうか、すなわち読者が適当なモジュールを探し出して、適当なときにそれを役立てられると思ってよいのかどうか、ということである。そうした判断を下すときの材料となる要因も類似している。繰返し部分がかなり長い場合には、それを各モジュールごとに繰り返すのは好ましくない。また、何度もその説明部分を呼び出したり、参照しなくてはならないとなると、読み進むプロセスが複雑になり、難しくなる。それはたび重なる呼出しが、コンピュータ・プログラムを複雑にするのと同じである。

●redundancyは何の役に立つか

redundancyは、メンテナンスを複雑にし、役に立たず、無駄であるように見えるが、マニュアルの中の読飛ばし、枝分かれ、逆戻りの数を減らしてくれる。マニュアルを読み慣れていない読者には、redundancyがあるかないかで、読みやすいか、読みにくいかが違ってくる。また反対に、複雑なマニュアルを読みこなすことのできる熟練ユーザー向けのものには、あまりredundancyを入れない方が無難であると、マニュアル作成者に気付かせることになるかもしれない。

（redundancyが多すぎると煩雑になることがある。指示があまり頻繁に繰り返されると、説教のように感じられる。たとえば、本書に見られるような繰返しの数は、プロの読者向けのテキストよりも、ユーザー・マニュアルに適している。先にも触れたが、これは私の勧める方法を実証したいがためである。）

モジュール間のredundancyについては、「まったく同じ」繰返しを用いるべきである。たとえば繰り返される段落や図をワープロに登録しておけば、正確にコピーできるようになるし、また1ヶ所を変更すると、それを繰り返すところすべてを変更できるようにもなる。

逆に、モジュール内でのredundancyは、繰返しを少なくして、等価値の説明を増やすとよい。2ページ（見開き1ページ）のモジュールのほとんどが、「3つの」redundancyのブロックを持っている。つまり、ヘッドライン／要約、図表、そして詳しい記述の本文－これらすべては、互いに補強し、重なり合っている。

8.10　枝分かれと階層構造を操作する

独立した個々のモジュールを順番に結び合わせたものとしてマニュアルを設計することは、非論理的だろうか？　枝分かれする部分や階層構造に関しては、どう扱えばよいか？　あるいはまた、1つのモジュールに入り切らない長くて複雑なプロセスに関してはどうだろうか？

●モジュールを階層化する

ある一定の因果関係にもとづく伝達方法－概念や説明が同一階層（レベル）上に並び、段組みも従属関係もないーはもっとも明快で、もっとも効果的である。短いトレーニング・ガイド、マーケティング用のパンフレット、20分ほどの映像などによるちょっとした報告、簡単にまとめた提案書－このような短く簡潔な伝達方法は、議論がなかなか進展しなかったり、売上げが頭打ちになっていたりする状態を一気に打開するためのはずみ車となったとき、もっともその効力を発揮する。マニュアルもこのように作ることができるはずであるから、こうした伝達方法同様、もっと効果的になるはずである。

しかし、膨大なページを要する表や定義や処理手順(枝分かれや逆戻りのあるプロセス)などを含む、より複雑なテクニカル・マニュアルやリファレンス・マニュアルに関してはどうか？

まったく問題はない。必要に応じて各ヘッドラインの頭を引っ込めたり、階層が分かるように番号をふったりといった簡単な方法で、階層構造は残したまま、読者に**より明確なものにする**ことができる。たとえば、あるモジュールが前のモジュールの“レベル”より上にあるか、下にあるか、同じであるかは、そのモジュールのヘディングを見るだけでも明確になる。また、そのモジュールの最初に出てくる情報を見ると、そのモジュールの“1つ上のレベル”は何か、ということが分かる－これは従来のマニュアルでは、目次を見なければどこにも出てこないものである。(たとえばあなたは今、第8章という大見出しの、1つ下のレベルのモジュールを読んでいることになる。)モジュール化されたマニュアルは、「それぞれのモジュールの中で」、そのマニュアルの階層と構成を見せてくれるのだ！

図表8.10にあるように、モジュール化されたマニュアルの目次は、連続型か階層型のどちらかである。左側の階層型のものは、2段階式で段組み処理を行なっている。右側のものは3段階式で、おなじみの小数点方式の表記を行なっている。実際、モジュール化の手法では、1つ1つの操作手順をあまり長くならないよう制限することによって、階層構造が「必然的に出来上がる」ことが多い。

●枝分かれをうまく処理する

モジュール化の手法では、操作手順が途中で「枝分かれ」していく場合にも、うまく説明することができる。枝分かれとは、オペレーターの選択によって、あるいはシステムがそのときの都合に応じて、2つ以上の代替経路のうちの1つに向かう行為である。従来のマニュアルでは、こうした枝分かれは、作成の過程において、もっとも複雑で油断のならない瞬間である。モジュール化されたマニュアルでは、最良の対処方法は、1つのモジュールの中に、代替経路を含んだ操作手順全体を詰め込んでしまうことである。1つのモジュールにうまく入らない場合は、モジュールの「終わりを枝分かれする地点に」持ってくるのがよいだろう。つまり読者は、次の代替モジュールの一方に進んで、もとのところに戻る必要がなくなるのだ。それは読まなくてよい何ページかの存在を意味し、並列の経路についての redundancy を意味する。

マニュアルの設計を構造化すれば、ライターは、マニュアルを報告書や提案書として扱うようになるだろう。なぜならそこでは、各モジュールがもっとも効果的で使いやすい順番に並ん

でいるからである。

連続型(単一階層)アウトライン

添付ディスクをコピーする
システムにハードの機種を登録する
オプションその他を選択する
住所録を準備する
住所録へデータを入力する
住所録のデータを変更する
他の住所録の一部から新しい住所録を作る
住所録全体を印刷する
住所録の一部を印刷する
封筒を印刷する
ラベルを印刷する
トラブル対策チャート

階層型(２階層)アウトライン

スタートの４つのステップ
　添付ディスクをコピーする
　システムにハードの機種を登録する
　オプションその他を選択する
　住所録を準備する
住所を入力する３つの方法
　住所録への最初のデータ入力
　住所録の旧データの変更
　他の住所録の一部から新しい住所録を作る
住所録を印刷する
　印刷すべき住所を選択する
　封筒を印刷する
　ラベルを印刷する
トラブル対策チャート

階層型(３階層)アウトライン

1.　スタートの４つのステップ
　1.1　添付ディスクをコピーする
　1.2　オプションと初期値を設定する
　　1.2.1　システムにハードの機種を登録する
　　1.2.2　オプションその他を選択する
　　1.2.3　住所録を準備する
2.　住所録を作るさまざまな方法
　2.1　キーボードからデータを入力する
　　2.1.1　住所録への最初のデータ入力
　　2.1.2　住所録の旧データの変更
　2.2　既存ファイルのデータを利用する
　　2.2.1　他の住所録の一部から新しい住所録を作る
　　2.2.2　他のデータベース内の住所録を利用する
3.　住所録を印刷する
　3.1　印刷すべき住所を選択する
　3.2　封筒を印刷する
　3.3　ラベルを印刷する
付録：トラブル対策チャート

図表 8.10

第9章

執筆・編纂:草稿を作成する

9.1 “GOTO”なき設計の有効性

GOTO なきマニュアルから利益を得るのは、まず第一に読者である。しかし、マニュアルを作成する側にとっても、GOTO がないということの意義は大きい。GOTO なき設計によって、モジュール相互間の独立性が保証され、そのためモジュールをどのような順番で書くことも、“執筆者”をどのように割りあてることも可能になるのである。それだけではなく、モジュールをその相互関係から切り離して、個々に見直し、テストすることもできるようになる。

●GOTOなきマニュアルとは何か

前述したように、“設計を凍結させる”とは、マニュアルの設計をもはや変えることが「できない」という意味ではない。設計とは、既定のルールを前提としており、したがって、その設計を変更する場合には、このルールに準じた一定の手順で行なわなければならないということを意味する。どんな小さな変更であろうと、「**GOTO なき設計**」を損ない、その変更作業の途中で、モジュール相互の独立性をおびやかすようなものであってはならない。

GOTO なきマニュアルとは、モジュールの集合体であり、モジュール相互間で考え得るあらゆるつながり－参照関係－は、そうした集合体の設計案の中で確認することができる。目的のモジュールの内部へと、うまく読者を導き得ないとしたら、その原因を突きとめなければならないのは、ほかでもない、その問題のモジュールを書いているライター本人である。Aというライターが、Bというライターのモジュールについて知っておかねばならないことは、すでに「ストーリーボード」の中、「モジュール・スペック」の中に示されている。

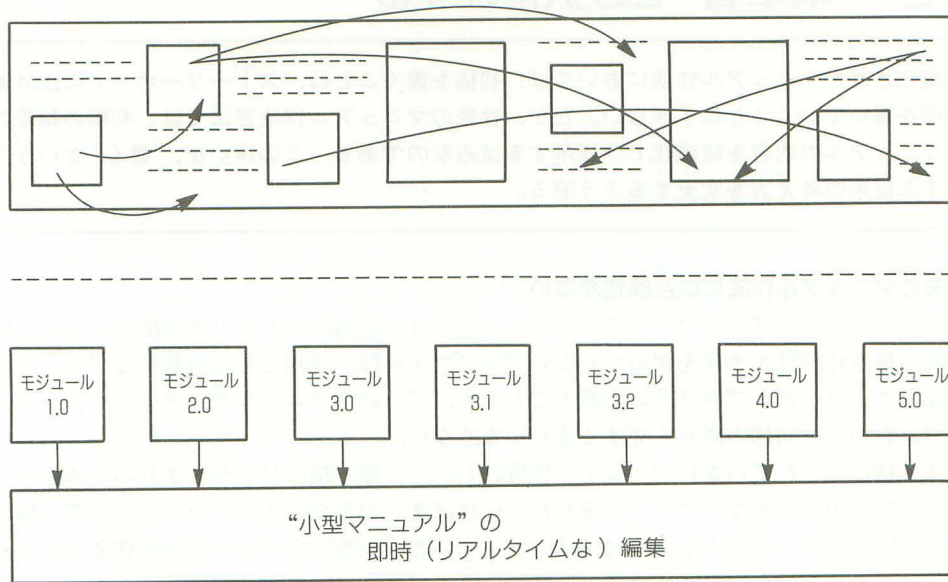
しかし、このモジュール相互の機能の独立性は、もし誰かがストーリーボード作業をやり直さずに、もとの設計から逸脱したり、もとの設計に追加したりすると、一瞬のうちに崩壊してしまう。(1個のモジュールの枠内に完全に収まり、他のモジュールに影響を与えないような変更については、もちろん何の問題もない。一般的にいって、ヘッドラインや要約部分に変更を必要としないようなモジュールの変更は、ライターの裁量に任される。)

●モジュールは自己完結している

さらに、もし**GOTO なき**ロジックを徹底できれば、プロジェクトのマネジャーは、ライターを集めるだけ集めて、そのパワーを最大限に活かすことができる。たとえば、1つのモジュールに取り組む時間(2～3時間)しか持っていないライターには、1つのモジュールだけを割りあて、仕事が遅れているライターの分は、他のライターに回したりすることが可能になるのである。

“執筆者”は、自分に割りあてられたモジュールのうち、どれから書き始めてもよい。1つのモジュールの穴埋め作業が遅れたとしても、他のモジュールは独立して書くことができ、互いに影響をおよぼし合うことがないからである。

GOTO なき設計の有効性は、初稿執筆のときばかりでなく、作成されたモジュールの見直しと編集の際にも、さらに如実に現われる。簡単にいえば、ストーリーボードの内容さえ分かっていれば、どのモジュールに対しても、見直しや編集作業を行なうのに不都合はない。さらに、設計を何ら変更しないと分かっていれば、ページ数や図の番号を、モジュールの書かれる順序に関係なく、前もって決定することができる。

ライターが書くのは、長くて複雑に入り組んだ一連の節ではなく、小型の自己完結的なマニュアルの集合体である。この自己完結型モジュールは、技術面での記載事項に関しては、すでに点検済みである。またそのマニュアルのロジックに適合しているだけではなく、そのマニュアルの「物理的な」書式にもフィットするものとなっている。したがって、ライターの

図表 9.1 <GOTOなきマニュアルの有効性>

仕事は、「何かを創造することではなく、既定の事柄を実行すること」なのである。

●構造化の手法の醍醐味は設計にある

このように、構造化された方法でマニュアル作成を行なうと、初稿を書き上げることのおもしろ味が薄れてしまう、という現象が起こってくる。本来なら、第一稿を書くことこそ、マニュアル作成においてもっとも複雑かつ困難で、しかも興味のある部分なのだが、構造化された方法の場合には、もっともつまらない過程となってしまうのである。（知力とセンスを必要とする事柄は、ほとんど設計の段階に移されたことを思い出していただきたい。）もちろん、1人の人間がマニュアルを書き上げる場合も、構造化された方法から得る利益に変わりはないが、プロのライターは、おうおうにしてみずからの設計をただ実行するだけでは満足せず、わき道にそれて、ところどころ自分の創意を入れたがる。

したがって、設計は技術の専門家とマニュアル作成者（あるいは編集者）の2人に任せ、残りの仕事は、この2人が割りあてた者に遂行させるというのが、ベストの方法といえよう。

9.2 “執筆者”を選択管理する

構造化されたマニュアル作成においては、初稿を書くことは、ストーリーボードの設計通りに細部を書いていくことにすぎない。だが、従来のマニュアル作成方法では、初稿の執筆こそが、マニュアルの内容を組織化し、提示する試みなのである。この違いが、“書く”ということに対する従来の考え方を変更するよう迫る。

●従来のマニュアル作成には互換性がない

マニュアル作成をプログラミングになぞらえるのは、必然的でもあり教訓的でもある。構造化分析と構造化設計の手法を用いていないプログラマーは、漠然とした直観的な“スペック”にたよって、ソース・プログラムを組んでしまう。プロのテクニカルライターでも、マニュアル作成において、同様の誤りを犯すことが少なくない。

従来の構造化されていないマニュアル作成において、第一稿はひとかたまりの全体である。せいぜい2、3の大まかなブロックに分かれていればましな方である。したがって、第一稿の作成は、ただ1人のライターの仕事となるか、せいぜいごく少数のライターの共同作業となるにすぎない。こうしたマニュアル作成では、アウトラインのスペック（仕様書）などないに等しいから、マニュアルの各部分をうまく結び合わせるという複雑な作業が、どうしても1人の人間に任されてしまう。外側からマニュアルをコントロールできないため、マニュアルの各部分のつじつまが合うようにするのは、内側からのコントロール、つまり1人のライターの頭の中で、ということにならざるを得ない。

このような方法では、マニュアルがいくつかの大きなブロック（それぞれが独立したマニュアルといってもいいようなもの）に分かれていない限り、チームワークは不可能に近い。作業を小さく分けてしまうと、各部分が互いにうまく適合することは望めなくなってしまう。この問題は、構造化設計の手法が導入される以前のプログラミングにつきまとっていた“互換性のないコーディング・スタイル”の問題に似ている。

●モジュールの数だけライターを集める

これと対照的に、各モジュールのスペックがすべてそろっていて、しかもどのモジュールも小型で独立したものであり、さらに、それぞれのスペックに技術的な内容に関する重要な記述がきちんとなされていれば、初稿の作成もまったく異なった様相を帯びてくる。さらに、各モジュールの並び方が、**GOTO**なきロジックにしたがわぬものであり、マニュアルの設計が「凍結されて」いれば、初稿の執筆は、通常“書く”という言葉から連想されるものとは似ても似つかないものになる。

この場合の“マニュアル作成”は、何人かの“執筆者”が、テキストや図表のまだ記述されていない細部を埋めていく作業から始められる。ただし、「1回の作業の対象となるのは、1つのモジュールだけ」である。したがって理論的には、モジュールの数と同じ数のライターが、それぞれ独立して作業することができる。少なくとも理論上は、どんなに長いマニュアルであろうと、必要な数の“執筆者”がそろっていて、ストーリーボードが完成していれば2～3時間の間に初稿を書き上げることもできるのである！

1人か2人の執筆者しか見つからない場合でも、マニュアル作成をモジュール化することの利点は大きい。マニュアル全体をいくつかの小単位に分割して、モジュール相互の関連性（“インターフェイス”）には気を遣わず、どのような順に書いてもよいことになるからである。

●書きたがらない人に書かせる

構造化の手法の大きな利点は、「書く」ことがいやでいやでたまらない人でも、ライターとしてどんどん起用できる点にある。こうしたやり方をする場合、執筆者として起用するのは、モジュールの細部の細部に至るまでよく知っている人である。執筆者の役割は、前もって指定された場所に、詳しい内容を正しく書き込んでいくことである。それらの内容は、関係者によってすでに設計され、点検、承認されたものである。このようにお膳立てされ、しかも“文章力は問題でない”といわれても、なお書こうとしないプログラマーやエンジニアがいたとしたら、彼または彼女は、どんな条件下でも、たぶん書こうとはしないだろう。

このような、およそ執筆者のタイプでないような人を、執筆者として起用することによって、マニュアル作成のスピードが上がるだけではなく、その技術的内容がより正確なものになる。書くのは苦手だが、専門知識は誰よりも確かであるという人に書かせると、ぎこちない文体であっても、正確さが増すのである。これはプロのライターにとっても都合がよい。なぜならライターは、初稿の修正と手直しに専念できるようになるからである。

モジュール番号	締切	担当執筆者
1..0	EHW 10/1	Brown
2.0	EHW 10/1	Brown
2.1	EHW 10/1	Brown
2.2	EHW 10/1	Brown
2.2.1	EHW 10/5	Lopez *
2.2.2	EHW 10/5	Blum *
2.3	EHW 10/1	Gilles
2.3.1	EHW 10/3	Gilles
2.3.2	EHW 10/3	Gilles
2.4	EHW 10/1	Wright *
2.5	EHW 10/1	Finch
3.0	EHW 10/1	Jones
3.1	EHW 10/1	Jones
3.2	EHW 10/1	Jones
3.3	EHW 10/5	Calder *
4.0	EHW 10/1	Finch
4.1	EHW 10/1	Finch
4.1.1	EHW 10/5	Finch
4.1.2	EHW 10/5	Warnier *
4.1.3	EHW 10/5	Jackson *
4.2	EHW 10/15	Archive
4.3	EHW 10/15	Archive
4.4	EHW 10/15	Archive

図表 9.2 ＜執筆者にモジュールを割りあてる＞

9.3 執筆者の管理にプロジェクト・マネジメントを応用する

マニュアルをたった1人のライターが書き上げる場合でさえ、ストーリーボードの作成やモジュール化による設計手法は大きな助けになる。さらにそのメリットは、プロジェクトの規模が大きくなり、チームのメンバーが増えるにつれて飛躍的に大きくなる。構造化の手法を用いると、「執筆作業の管理面」に劇的な変化がもたらされる。まず第一に、執筆作業の全段階を通じて、力配分が均一化され、第二に、プロジェクト・マネジメントのテクニックを、そっくりそのままマニュアル作成に応用できるということである。

●従来のマニュアル作成ではプロジェクト管理ができない

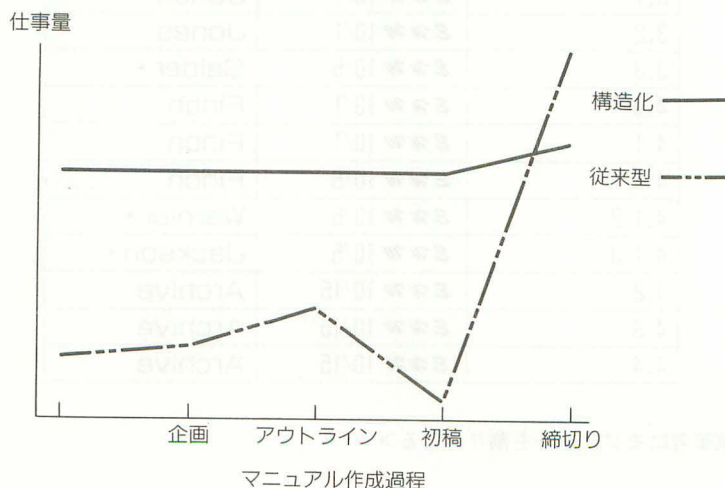
マニュアル作成を構造化することにより、初稿を執筆・編纂するやり方が根本的に変わり、マニュアル作成者は、いわばマネジャー（管理者）の役割を果たすようになる。

少なくとも2人以上のライターを必要とする規模のマニュアルなら、そのうち誰か1人が、全体を統括する責任者とならねばならない。しかし、この責任者としてプログラマーの1人に白羽の矢を立てたとしても、また、マニュアル作成の管理のプロを連れてきたとしても、従来のマニュアル作成の体制では、まともな管理などできるはずがない。従来のアウトラインをもとに仕事を進める限り、マニュアル作成にかかる時間とコストをコントロールすることは、ほとんど不可能であり、品質管理にも限りがある。

図表9.3aは、従来の方法と構造化の方法を比較したものである。従来のやり方では、アウトラインの作成とプランニングに少し時間をかけた後、仕事量の低迷期が長く続く。この間、プログラマーやライターは、たいてい守備範囲とサイズのはっきり決まっていない執筆の分担を割りあてられ、呻吟に呻吟を重ね、ついに締切りを迎え（あるいはオーバーし）てしまう。責任者側では、やきもきしながら初稿が上がってくるのを待っているが、予定通りできてくることはめったにない。

したがって、責任者自身が、ほとんどの部分を大急ぎで書き上げざるを得ないはめになり、編集と見直しのための時間など、望むべくもなくなってしまう。

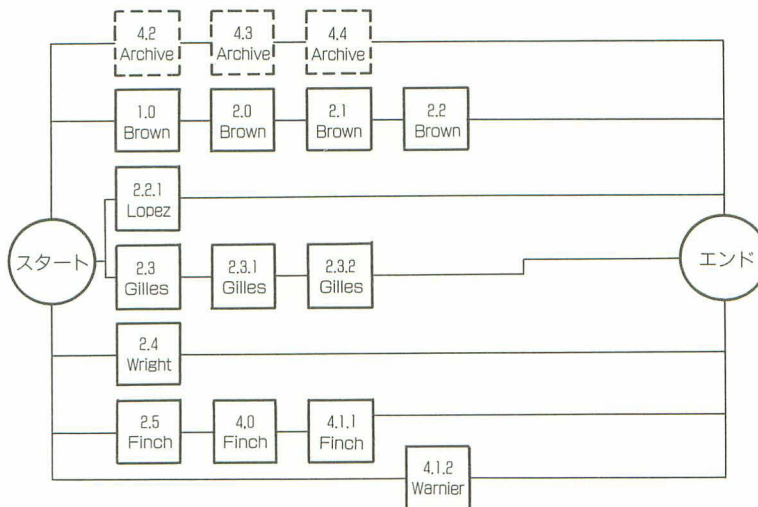
●構造化の手法ではコストやスケジュールを管理できる

従来型のマニュアル作成とは対照的に、構造化されたマニュアル作成においては、プラン

図表 9.3a ＜従来型のマニュアルと構造化マニュアルの作成過程における仕事量の対比＞

ニングと設計に時間がかけられる。その代わり、アウトラインが完成していれば、数時間の間にモジュール（すべて小型で独立している）の第一稿を責任者のもとに集めることができる。したがって、編集、テスト、訂正などに要する時間をも含め、全過程を通じて、作業の力配分が「均一化」されるわけである。

予定通りに割りあてのモジュールを書き上げることのできない執筆者がいれば、責任者はただちにその原因を調べ、必要ならば、時間的余裕のあるうちに、他の執筆者を割りあてることができる。

このように、マニュアル作成を分担作業にし、そうした作業のネットワーク全体を通して細かい調整を行なっていくと、構造化の方法のもう1つのメリットが見えてくる。各モジュールは、はっきりと定義された1つの作業単位であるため、プロジェクトのネットワークの中に正確に位置付けることができるというメリットである。各モジュールは、仕事量のはっきりした1つの作業であり、担当者、推定所要時間はもとより、場合によっては、アートワーク（レイアウト、デザインなど）や製作（版下製作、印刷、製本など）にかかる費用もはっきりしている。管理者は、コストを前もって見積り、モジュールの割りあてをやりくりすることにより、完成時期を予想し、調整することができる。万事がうまくいった場合、モジュールはすべて相互に独立しているため、作業工程を制約するのは、1人の執筆者に複数のモジュールが割りあてられている場合だけになる。

モジュール化の手法を徹底すればするほど、予算を守り、作成プランの“クリティカルパス*”を短縮することができる。さらにライターを管理する側（素人であれプロであれ）にとって、現在、小型コンピュータで可能となっている新しいプロジェクト・マネジメント・ツールを利用するチャンスも出てくる。

図表 9.3b ＜マニュアル作成作業のネットワーク＞

訳注＊クリティカルパス：Critical Path。プロジェクト・マネジメントの用語で、スケジュールに遅れが出ると、プロジェクト全体に影響を与える一連の作業の筋道。個々の作業の所要日数と優先順位を分析すると、プロジェクトを順調に運ぶためにどうしても遅延させることのできないひと連なりの作業の筋道が、少なくとも１本はあることが分かる。並行して行なえる作業を増やすことによって、このクリティカルパスを短縮することができる。

第10章

編集：読みやすさ、明確さのために テストと見直しをする

10.1　初稿をテストする：中心課題

初稿をテストする目的は、表現上の誤りを“取り除く”ことである。読者のミスを誘発するような文章、あるいは難解で曖昧なために、読み直さなくてはならないような文章などを、できる限り何回も修正することである。

●編集作業は言葉の見直しから

“構造化されていない”マニュアルの初稿は、従来型のアウトラインや仕様書によって書かれているため、たいていの場合、かなり読みにくく、新しい技術的な内容がぎっしり詰まっているのが通常である。したがって、技術面での見直しが早急に必要となる。そして時間が許せば、文体の見直しを行なって、さらに他の技術面での見直しをすばやく行なうことになるだろう。一方、構造化されたマニュアルにおいては、モジュールごとの見直しが可能である。構造化されたマニュアルの技術的な内容は、すでにチェックを受けているため、言葉と体裁をまず整え、それから細部に記載漏れがないかどうかについて、技術面での見直しを簡単に行なうだけでよい。

構造化されていないマニュアルの初稿は、前もって責任者からチェックやテストをまったく受けていないため、ほぼ90％が新しい内容に関する記述となる。このような初稿に対しては、技術面で正確かどうか、細かいところをチェックする作業から始めるしかない。ところが、初稿が編集されておらず、散文的なために、この見直し作業は難航する。したがって言葉や図表を整え、読みやすくし、より効果的なものにするといった作業を行なう機会は、事実上なくなってしまう。

これとは反対に、構造化されたマニュアルの各モジュールは、それ自体で小型の自己完結型マニュアルとして機能する。モジュールの技術的な内容と、モジュール間の論理的なつながり

図表 10.1　＜初稿テストの２つの方法＞

構造化されていない初稿の場合	構造化された草稿の場合
第一稿全体について	各モジュールごとに
「まずは技術的な見直しから」 ・複雑なテキストを初めて読む ・主な誤りを見つける ・技術的誤りを、混乱した、曖昧な言葉遣いの中から識別	「まずは言葉の見直しから」 ・手の加えられていない原稿の言葉の見直し ・曖昧な内容を明確にする ・欠陥と矛盾を見極める
「言葉の見直しは駆け足で」 ・ケアレスミスのぞんざいな見直し ・些細な編集見直し ・印刷と製作のための仕様書を書く	「１回目の技術的な見直し」 ・ストーリーボード上でまだテストされていない技術的内容の確認 ・最後になされた技術的変更との整合性を見る
「最後は技術的な見直し」 ・あわただしい、形式的な二重チェック ・役に立たない更新(間違いだらけのページ)	「最後は言葉の見直し」 ・言葉と図表の有用性をさらに向上させる ・製作、校正を注意深く行なう

は、あらかじめ技術面の専門家によって明確に定められており、また、他の専門家によってチェックを受けているのである。モジュール内の記述が技術的に見て突飛であることはなく、また、見直し作業が最後まで引き延ばされることがないため、**言葉や技巧の面での見直しから開始することが賢明である。技術面での誤りを見つけるには、初稿に対してよりも、整理された（編集された）文章に対しての方が容易である。**

言葉の見直しは、テキストの「明快さ」（曖昧な情報や誤解を招くような情報が取り除かれているかどうか）、「読みやすさ」（必要以上に難解ではないか、特に対象読者にとって読みづらくないかどうか）、といったことを目標に行なわれる。一般的には、初稿の編集に努力を傾注すればするほど、意味が一層はっきりし、読みやすくなる。

たとえプロのライターによって書かれた原稿であっても、マニュアルを使いやすくするためには、初稿に対して例外なく編集作業を行なわなければならない。

●見直しの５つの観点

編集作業を行なう場合、通常以下の５つのカテゴリーにおいて見直しが行なわれる。

・**機械的作業**：文法上のミス、誤字・脱字、不正確な区読点を訂正する。これは意味の混乱を正す基本的な作業である。
・**適切な言葉に直す作業**：読者になじみの薄い言葉を使って必要以上に難しくなっている文章、調子の悪い言い回しなどを修正する。
・**意味をはっきりさせる作業**：複数の意味に解釈される、あるいは読者の誤解を招くような単語、文節、文章、構文、図表を修正する。
・**読者のなじみやすさを高める作業**：たどたどしい、あるいはくどい表現、難解な図表など“必要以上に飾りたてて”、結局は読者にとって使いにくくなってしまうようなものを修正する。読者がもう一度文頭に戻らなければ意味がはっきりしない文章など、初稿には必ずといっていいほど存在する言い回しを修正する。
・**読者を引き付ける作業**：言葉遣いに注意し、緻密な編集を行ない、文の長さに工夫をこらし、巧みな作図を行なうなどの作業を通して、マニュアルに対する読者の興味を高め、魅力的な内容にする。

もし、会社の誰かが編集に費やされる時間にクレームを付けたとしたら、こういうといい。「意味の明らかでない文は、技術的なミスを覆い隠し、トラブルを招く」と。たとえば次のような文はどうか。

「管理陣には、プロジェクトの継続を承認するために、書式A51にサインすることが求められる。」

この文はとんでもない混乱を招く。この文のもっとも悪い点は、管理陣が、ある書式へのサインを要請されているとも、あるいは強制されているとも解釈できることである。これらの解釈は、本来の意味とは異なる。後述する原理にあてはめるなら、以下の通り改めるべきであろう。

「管理陣は、プロジェクトの継続を望むならば、書式A51にサインしなければならない。」

10.2　単語や言い回しのバグを取る

もっとも簡単な編集作業は、よく使われる単語や句のバグを取り除いたり訂正したりすることである。特に、長たらしい言葉や流行の言葉で飾りたてられた表現、必要以上に言葉を使っている表現、逆に言葉を省略しすぎた表現、語や句を間違った場所に置いている表現、などには注意が必要である。

言い回しの上でよく犯されるミスの中には、二重の弊害をもたらすものがある。第一はテキストの中の技術的な内容の誤りや脱落を隠蔽してしまうことである。第二は、読者がテキストを読み返したり読み違えたりする可能性を増やしてしまうことである。

●表現の凝りすぎ

日常的な短い言葉で充分なのに、次のような持って回った表現で文章を飾ることがある。

「使う(use)」の代わりに「～を利用する(utilize)」
「助ける(help)」の代わりに「助成する(facilitate)」
「開始(start)」の代わりに「創始(initiation)」
「押す(press)」の代わりに「押し下げる(depress)」

誤解しないで欲しい。短い単語を使うこと自体に意味があるわけではない。また、技術的に正しい言葉を、不正確な短い言葉に直してもメリットはない。しかし、次のような言い換えにも何らメリットはない。

「示す(show)」の代わりに「指し示す(indicate)」
「送る(send)」の代わりに「普及させる(disseminate)」
「させる(cause)」の代わりに「引き起こす(effectuate)」

文章を飾りたてる方法として、次のような流行語(仲間内の専門用語)を使う場合もある。

「能力(ability)」の代わりに「可能出力(capability)」
「ランク付けする(rank)」の代わりに「優先順位を付ける(prioritize)」

また「環境(environment)」という言葉は、あまりにも頻繁に、そしてあまりに不注意に用いられているため、同じマニュアルの中でも、反対の意味に使われていることさえある。さらに、透過的(transparent)という言葉にも注意して欲しい。この言葉は、商業英語では“明白な”という意味だが、コンピュータ業界では“目に見えない”という意味になる。

●言葉の使いすぎ

必要以上に多くの単語を使うことは、300ワード(日本語で700字)で説明できる考えを、1000ワード(日本語で2500字)に増やすという出来の悪い学生の方法である。以下に若干の例を掲げる。

・～というケースに合致すると立証できるならば＝もし～なら(should it prove to be the case that = if)
・～の使用を行なうという意味によって＝～で(by means of the utilisation of = with)
・～をする時間の早い時点において＝そのとき(at the ealier point in time = then)
・見積りの算出を行なう＝見積る(perform the calculation of the projection = project)

初稿では表現がどうしても冗漫になる。冗漫な表現でもっとも目につくのは、

○動詞の名詞化：(「区別する(distinguish)」の代わりに「区別を行なう(make a distinction)」、「～を結合する(link)」の代わりに「～間の結合を実行する(accomplish linkage between)」)

○単純な表現の複雑化：(「～のために(to)」の代わりに「～を目的として(in order to)」、「～について(about)」の代わりに「～という事柄に視点を向けると(with regard to the subject of)」)

という2つである。また場合によっては、句の代わりに節を用いることで表現が冗漫になる。

たとえば、「テスト計画を承認して、われわれは～を始めた(Having approved the test plan, we began the...)」の代わりに「われわれはテスト計画を承認した後に、われわれは～を始めた（After we had approved the test plan, we began the...)」といった具合である。

●省略のしすぎ

必要以上の言葉の省略が、エンジニアやプログラマーの文章によく見うけられる。ライターにも、簡潔さを求めるあまり、次のように理解不能になるまで句や節を縮める者がいる。

「このシステムには"自然言語型報告書作成機能"がある(English-like report generating capability)」
「隣接セクター・リファレンス・デジグネイター(contiguous sector reference designators)」
「操作可能計画策定用資料書式設計基準(operational planning materials format design criteria)」
「管理責任分担ヒストリー・ファイル(management resposibility assignment history file)」

このような名詞の結び付き方（プログラマーなら「名詞結合形態」というだろうが）について理解できる人はまずいない。ちょっとやそっとの修正を加えても、役には立たないだろう。

上述のようにわけの分からない方法で前置詞を省略する者や、単語のすし詰めを作る者は、「その(the)」とか「ある(a)」といった"無用な"言葉を削除したがる。また、「上述の(that)」といわずにすむ場合は、彼らはそれを省略してしまう。そればかりか、必要でない句点(comma)はできるだけ省略したがる。こうした極端な省略指向や削除指向は、言葉の流れを破壊し、確かに短い表現ではあるが、解読するのは長くなるという文章を作り出してしまう。

●語順の間違い

単語や語句を置く位置を間違えることも、意味の流れをとぎれさせてしまう。英語では、修飾語は被修飾語と並べて（一般的には、直前に）置かれるべきである。しかし、初稿ではおうおうにして、修飾語の多くが置かれるべき所に置かれていない。特に、「唯一(only)」、「およそ(nearly)」、「ほとんど(almost)」、「すでに(already)」、「～でさえ(even)」、「ちょうど(just)」、といった言葉は、間違った場所に置かれることが多い。

「唯一、一般労働条件外の従業員に時給制度を導入せよ」(only enter the hourly rate for exempt employees.)

この文の「唯一(only)」は、何にかかっているのだろうか。「時給制度のみ(Only the hourly rate)」という意味か、「時給のみの制度(the hourly rate and nothing else)」という意味か、あるいは「一般労働条件外の従業員のみ(the hourly rate only for exempt employees.)」だろうか。

「このシステムは、ほとんどすべての人々のチェックをプリントします」(The system prints nearly everyone's checks.)

といいたいときに、

「このシステムは、すべての人々のチェックを、ほとんど(nearly)プリントします」
(The system nearly prints everyone's checks.)

と書いてはならない。

同様に、修飾部と被修飾部が遠くなってもならない。なぜなら、

「企業方針に抵触するので、すべての非合法アクセスを報告しなさい」
(Report every unauthorized access in keeping with company policy?)

という文では、

「ある非合法アクセスが企業方針に抵触するものでなかったら、報告しなくてもよい」
(if the unauthorized access is not in keeping with company policy, you should not report it)

という解釈も可能となるからである。

10.3　文章のバグを取る

文章をダメにする要素は星の数ほど存在するが、それらが読者を立ち往生させたり、注意力を散漫にしたりする場合、おおむね次の5つの欠陥が原因となっている。つまり、読返しを強いるような文章構造、つながりのよく分からない述語、主語の分かりにくい受動態、宙ぶらりんの導入部、牛のよだれ文がそれである。

●強調したい部分は文末に

文書を読みやすくする秘訣は、その文章で執筆者が“強調したい事柄”（強調したい事柄がある場合）を最後に持ってくることである。これ以上論理的な方法があるだろうか。文章が長い場合、その文章は頭から順々に読まれ、理解されていく。したがって、文章の最後の部分が、記憶にもっとも鮮明に残るのである。もし、どんな重要な事柄でも、文頭に置かれるなら、読者は文章を読み終えてから、最初に戻って“繰り返し”読まなければならないだろう。

素人のライターでも、練習さえすれば覚えられることだが、初稿をチェックする際に、強調されるべき事柄が文末にきているかどうか調べ、もし強調される事柄が文末になければ、文章を作り直し、強調部分を文末に持ってくることができる。ただし、技術的な理由から強調部を文末に持ってこられない場合は、この限りではない。では、この変換練習を始めてみよう。

変換前「コスト低下は、新しい作業手順の最大特質である。」
(Reduced cost is the main advantage of this new procedure.)
変換後「新しい作業手順の最大の特質は、コスト低下である。」
(The main advantage of this new procedure is reduced cost.)

同様に、命令文の編集を行なう際は、キーワードを最後に置けばよいのである。また、条件付きの指示文（if-then）では、「指示の部分（then）」を最後に持ってくるべきである。次に「良い例」と「悪い例」を挙げる。

[悪い例] DFILが入力される。(DFIL is typed.)
[良い例]入力すべきはDFILである。(Type DFIL.)
[悪い例] DFILと入力すると、ファイル名がチェックできる。
(Type DFIL to see what file names have been assigned.)
[良い例]登録されているファイル名をチェックする場合は、DFILと入力せよ。
(To see what file names have been assigned, type DFIL.)

●述部をお払い箱にするな

強調部が文末にくると、文章の中の“動作部分”は、たいてい「述部」にきて、主部にはほとんどこない。しかし、多くのライターは、強調部を文頭に持ってくるばかりでなく、関心をひく事柄を、すべて動詞を使う前にいってしまうことさえある。後に残るのは「つながりのよく分からない述部」である（エディターの中にはこれを“お払い箱の”述部と呼ぶ者もいる）。

「日本人によるダンピング（標準価格以下の値をつける→安く見積る）の可能性が存在している」という文を考えてみよう。この文の述部は“存在している”という部分である。しかしこの文を組み換えてみるとライターがわれわれに理解させようとしたことが、おのずと分かる。

(1)日本人がダンピングする恐れがあるのは、われわれである。(The japanese may underprice us.)
(2)われわれがダンピングされる恐れがあるのは、日本人によってである。(We may be underpriced by the japanese.)

(1)も(2)も文法的に正しい文だが、(1)は「われわれ」を強調しており、(2)は「日本人」を強調している。

●受動態の良い例悪い例

言語には、文頭から文末への言葉の置換えを可能とする多くの方法がある。「置換え方法」の中でも、もっとも使いやすいのが受動態である。次に例を見よう。

（1）ZAKO社が買収したのは、XTRONである。(ZAKO Industries acquired an XTRON.)
（2）XTRONが買収された相手は、ZAKO社である。(An XTRON was acquired by ZAKO Industries.)

このように能動態から受動態に換えることで、強調される言葉も変わってくる。

一方、多くの編集者やライター養成者は、素人のライターに対して、受動態に注意するよう警告している。これにはそれなりの意味がある。受動態にしたばかりに、せっかく分かりやすかった文脈を破壊し、かえってもつれさせてしまうこともある。以下にその例を見よう。

[受動態]柔軟性の乏しさが露呈されるのは、このシステムによってである。
(Insufficient flexibility is exhibited by the system.)
[能動態]このシステムは、あまりに柔軟性に乏しい。(The system is too inflexible.)
[受動態]安価な照合作業と結合作業が実現されるのは、このデバイスによってである。
(Cheap collating and binding are accomplished by this device.)
[能動態]このデバイスは照合作業と結合作業を安価に行なう。(This device collates and binds cheaply.)

受動態では表現がくどくなり、分かりにくくなる。しかし受動態は、扱いさえ気を付ければ、もっとも“強調したい事柄”をもっとも効果の発揮できる位置（つまり文末）に持ってくることができる。

●導入部と主部の主語に注意

また、主要な部分を文末に置くもう１つの方法として、「導入部」を利用する方法がある（ほとんどの条件付き指示文が導入部を持っている）。この方法の危険なところは、導入部が宙ぶらりんになるという点である。つまり、単語の意味が文の主部と切り離されてしまうのである。

これを克服することも難しいことではない。導入部を文法上の主語と結びつけ、主語を句点の直後に置くのである。以下に良い例と悪い例を掲げる。

[悪い例]あなたが賃金台帳を簡単にまとめたいとお望みなら、PAAYはあなた向けのシステムです。
(With your simple payroll requirements, PAAY is the system for you.)
[良い例]あなたが賃金台帳を簡単にまとめたいとお望みなら、「あなた」はPAAYシステムを使うべきです。
(With your simple payroll requirements, *you* should use the PAAY system.)
[悪い例]コマンドの意味をすばやく見るために、用語一覧はワークステーションごとに置かれています。
(To locate definitions quickly, glossaries are posted at each work station.)
[良い例]コマンドの意味をすばやく見るために、「オペレーター」はワークステーションごとに置かれた用語一覧を利用することができます。
(To locate definitions quickly, *operators* can use the glossaries posted at each work station.)
[悪い例]コールドスタートでシステムを立ち上げるときは、OSテープがロードされます。
(When coldstarting the system, the operating system tape is loaded.)
[良い例]コールドスタートでシステムを立ち上げるときは、(「あなた」は) OSテープをロードしてください。
(When coldstarting the system, (*you*) load the operating system tape)

●牛のよだれ文は禁物

ほかにも、文末で意味のつながりが切れてしまうこともある。“プリンターの保守作業禁止は喫煙時”(Do not service the printers while somokihg）といったバカ気た表現には、注意して欲しい。（この文は“煙の出ている間はプリンターの保守作業禁止”とも読める）

最後に、肝に命じておかなければならないのは、文章というものは「牛のよだれ」のようにだらだらと続けるべきものではないということだ。たとえば、次のような文書は手の付けようがない。

実線用、破線用、ぼかし用、センターライン用、不可視ライン用フォントに加えて、さまざまなポイント数の活字セットが発売され、可変スペーシング(幅)、挿入オプションのレイヤー、右寄せ、左寄せ、センタリング付きでセンターライン用の新製品となりました。(In addition to solid, dashed phantom, centerline, and invisible line fonts, numerous linestring fonts are available that provide generation about a centerline with variable spacing(width). layer of insertion options, and left, right, and center justifications.)

訳注：このモジュール内の訳文は、日本語としてやや不自然な語順になっているが、それは英文の［良い例］と［悪い例］を際立たせるためである。念のために。

10.4　指示文を曖昧にする10項目

単語、文節、文章のバグは、どれも例外なく「正確さ」と「有用性」を引き下げる。したがって曖昧な指示文に対しては、高い代償が支払われることになる。

1．流行語指向で遠回しな表現になっている

[編集前]インフォメーション・センター環境において、マネジャーは、優先化ランキングの方法を活用して、均等分割スケジューリングを助長するものとする。(In the Information Center environment, the manager should utilize a prioritization ranking to facilitate equitable scheduling.)

[編集後]インフォメーション・センターで、マネジャーは、すべての仕事に優先順位を付け、公正なスケジュールを作る。
(In the Information Center, the manager ranks each job to yield a fair schedule.)

2．必要以上の言葉が使われている。

[編集前]書込みの許可が与えられているのはどのファイルかを識別するための知識に不足があるという事態に陥った場合、PRIFIL コマンドの使用を実行せよ。(In the event that you have a lack of knowledge regarding which files you have permission to write in, make use of the PRIFIL command.)

[編集後]どのファイルに書き込んでよいか分からなければ、PRIFIL コマンドを使え。
(If you do not know which files you may write in, use the PRIFIL command.)

3．必要以上に言葉が省略されている

[編集前]カラム・ヘディング修正許可は、HCOL の入力で得られる。
(Column heading revision permission may be obtained by HCOL entry.)

[編集後]カラムのヘディングを変える許可を得るには、HCOL と入力せよ。
(To get permission to change the headings of the columns, enter HCOL.)

4．単語や文節が間違った位置に置かれている

[編集前]ワーク・シートには、変更ではなく、修正を書くだけにする。
(Only write corrections, not changes, on the worksheet.)

[編集後]ワークシートには、変更ではなく、修正だけを書け。(On the worksheet, write only corrections, not changes.)

5．文頭に戻らなければ文意がはっきりとしない表現がある

[編集前]「Clear Rest」キーを押すときは、カーソルの後ろにあるものすべてを消去するときである。
(Press the *Clear Rest* key if you want to erase everything after the cursor.)

[編集後]カーソルの後ろにあるものすべてを消去したいときは、「Clear Rest」キーを押せ。
(If you want to erase everything after the cursor, press the *Clear Rest* key.)

6．意味のはっきりしない述部がある

[編集前]キーパンチング開始前のコーディング・シートに対するスポット・チェックの有効性は、特質に値する。
(The efficiency of spot-checking the coding sheets before commencing keypunching is worthy of mention.)

[編集後]有効的なことは、キーパンチングを始める前にコーディング・シートをスポット・チェックすることである。
(It is efficient to spot-check the coding sheets before you start to keypunch.)

7．意味のはっきりしない受動態がある

[編集前]注意が払われなければならないのは、機密情報の送信である。(Care must be exercised in sending sensitive data.)

[編集後]機密情報の送信には注意せよ。(Send sensitive data carefully.)

8．関係のはっきりしない表現がある

[編集前]貸借勘定が一致したら、仕訳伝票ファイルは凍結されなければならない。
(When reconciling the account, the encumbrance file must be frozen.)

[編集後]貸借勘定が一致したら(あなたは)仕訳ファイルを凍結せよ。
(When reconciling the account, (you must) freeze the encumbrance file.)

9．関係ない人物が登場する

[編集前]オペレーターはその時点で自分の暗証コードを入力する。(The operator then enters his or her security status.)

[編集後]あなたの暗証コードを入力せよ。(Enter your security status.)

10．"義務付けを曖昧にする表現"がある

[編集前]オペレーターはデータを入力する前に、40時間の指導を受けることが1つの要請である。
(It is a requirement that operators receive 40 hours of instruction before they enter any real data.)

[編集後]オペレーターはデータを入力する前に、40時間の指導を受けなくてはならない。
(Operators must receive 40 hours of instruction. . .)

図表 10.4a 〈編集前の記述〉

1. あなたのハードウェア構成が充分な RAM 容量を有するならば、あなたはシステムのウィンドウ機能を活用できる。
(If your configuration has sufficient RAM capacity, you may utilize the system's windowing capability.)
2. 予測に対する評価を留保する場合をはっきりさせるならば、あなたには、これに代わるディスカウントレートを使用することについて選択権がある。
(Should it prove to be the case that you have some reservations regarding the forecasts, you have the option of using alternative discount rates.)
3. 早い時期のマニュアル設計は、構成上の有用性の利益を生む。
(Early manual design yields procedural usability benefits.)
4. スライド・メーカーが唯一利用され得るのは、512K 装備のシステムおよびハードディスクとともにである。
(The slide-maker only can be used by system with 512K memory and hard disks.)
5. PINSTALL と入力するのは、印刷のオプションを変えるときである。
(Type PINSTALL to change the printing options.)
6. 最低10分間に1回データをセーブするという必要性を満たすためには、あなたの注意がなくてはならない。
(The urgent need to save data at least every ten minutes is called to your attention.)
7. ファイル結合は、キー指定によって実現される。
(File linkage can be accomplished by key specification.)
8. 計算機能を呼び出すには、〈alt〉と〈c〉が押されなければならない。
(To call the Calculater, 〈alt〉 and 〈c〉 must be pressed.)
9. 事務員は、この時点で目的のファイル番号を入力すべきである。
(The clerk should then type the number of the desired file.)
10. 前の当番の人からのトラブル報告書を読むことは、交替したオペレーターの責務である。
(It is the responsibility of arriving operator to read the trouble report from the latest shift.)

図表 10.4b 〈編集後の記述〉

1. あなたのコンピュータに充分なメモリーがあるなら、ウィンドウ機能を利用することができる。
(If your computer has enough memory, you can use the window feature.)
2. 予測を疑うのなら、他のディスカウント・レートを試すことができる。
(If you doubt the forecasts, you may try other discount rates.)
3. 早い時期からマニュアルを執筆すると、使いやすい構成になる。
(Writing manuals early makes the procedures easier to use.)
4. スライド・メーカーは 512K 装備のシステム、あるいはハードディスクがないと利用できない。
(The slide-maker can be used only by systems with 512K memory or hard disk.)
5. 印刷のオプションを変えるには、PINSTALL と入力せよ。
(To change the printing options, type PINSTALL.)
6. 最低10分間に1回は、データをセーブせよ。
(You must save the data at least every ten minutes.)
7. ファイルを結合するには、キーを指定せよ。
(To link the files, specify the keys.)
8. 計算機能を呼び出すには、（あなたは）〈alt〉と〈c〉を押せ。
(To call the Calculator, (you) press 〈alt〉 and 〈c〉.)
9. 目的のファイル番号を入力せよ。
(Type the number of the file you want.)
10. 交替したオペレーターは、前の当番の人からのトラブル報告書を読まなくてはならない。
(The arriving operator must read the trouble report from the latest shift.)

10.5　読みやすさのために編集する

「可読性（文書の読みやすさ）」という言葉は、個々の文書の難解性と反対のことを指す。「難解性」という言葉は本書では文章を読み取るのに必要なあらゆる努力を示す。可読性を表わす多くの指標の中でもっとも一般的なのは、簡単な方法で文章の難解性を表わす指数を就学年数（学年）とほぼ一致させる、Robert Gunning 氏の「フォグ指数」と、Rudolph Flesch 氏の「読取り尺度」である。

今日では、「可読性」を測定するための多くの指数や尺度がある。最近では、５つの尺度で可読性を測る Bell Labs 氏の“ライターのワークベンチ”がある。

可読性の尺度と呼ばれるものは、どれも大ざっぱである。中にはでっち上げではないかと思われるものもある。その尺度では“読みやすい”と判定されるように巧みに作られた文章が、実際には、ほとんど読むに耐えないものであることを誰もが経験しているだろう。しかし、これらの尺度の目的は、読者が文意を読み取るときの難易度を、“客観的に”評価するところにある。もっとも一般的な尺度は、読解が困難になるにしたがって、数値が上昇する。そして、この数値と“読解に必要な就学年数（学年）”とが等しくなるように工夫されている。

もっとも有名なのは、Robert Gunning 氏の「フォグ指数」である。この指数は、１つのセンテンスの中の平均単語数に、“難しい”単語のパーセンテージを加え、就学年数と読解の難しさが一致するように、この和に定数0.4をかけるという方法を取っている（“難しい単語”とは３音節以上の単語である。ただし、固有名詞、簡単な単語の複合語、３音節であっても「-ed」か「-es」で終わる単語は、数えない。）

では、フォグ指数をテストするために、以下のテキストを見てみよう。このテキストは、世界最大級のソフトウェアやハードウェア会社によって作成されたものである。

Today's advancements in educational management combined with the rapid growth in student enrollment in schools has emphasized the need for data processors to be used in establishing and maintaining a student records data base, required for providing attendance and academic mark reporting data to satisfy several disciplines. The purpose of this program product is to provide a systematic procedure for recording, retrieving, manipulating, and reporting significant student data, such as attendance and academic mark information. One of the objectives of this program is to provide effective data on individual students as well as aggregate, statistical reports needed for sound analytical decisions by educators and administrators.

この文章には、センテンスが３つ、単語が105、Gunning 氏のいうところの“難しい”単語が34個ある。したがって、フォグ指数は、0.4×（35＋32）＝26.8となる。

これは、就学年数が26年以上ないと、たやすく上記の文章を読むことができない、ということを表わす。これでは、この文章を簡単に読める人は、地球上にはいないことになる！しかし、文章の内容は、ほとんどないに等しく、少し修正を加えれば、フォグ指数は10～11に低下するのだから、この文章の作成者にとって、まったく気の毒な結果としかいいようがない。

（米陸軍は、軍内で使用しているマニュアルの難易度を調べるのにもう１つ別の尺度を採用している。それは、Rudolph Flesch 氏の「読取り尺度」を軍自身が改良したものである。この改訂版 Flesch 尺度を上の文書にあてはめてみると、やはり結果が就学年数と一致するよう工夫されている。）

もちろん、このような文章の可読性の指数が低ければ、問題のすべてが解決されるわけでは

ない。レベルが6～7（年）では、知性に訴えるような表現ができないという場合もあり得よう。また、このような簡単な指数に対して、あるいは単純に可読性を測定するということ自体に対して、異論を唱える人もいよう。しかし問題は、非常に難解な文章から意味を読み取ることは、大部分の読者にとって不可能であるという否定できない事実である。しかも、バグが排除されたマニュアル、間違いのないマニュアル、GOTOを排除するように正しく企画・設計されたマニュアルであっても、依然マニュアルが読みにくいという問題を抱えている場合があるのである。

したがって、以下の基準を採用することが有用である。

技術書はすべてフォグ指数が14を超えてはならない。ビジネス文書は、すべて同12を超えてはならない。事務員向けマニュアルは、すべて同8を超えてはならない。技術書が簡単であればあるほと、多くの人々にとって読みやすいものとなるのである。

図表 10.5 <可読性の2つの測定方法>

フォグ指数(Gunning)：

読解に必要な就学年数＝0.4×(センテンスあたりの平均単語数＋難しい単語*のパーセンテージ)

＊難しい単語＝3音節以上の単語。ただし以下のものを除く。

- 固有名詞
- 短い単語の複合語
- 3音節目が -ed あるいは -es の単語

Flesch 読取り尺度　**(米陸軍による改訂版)**

読解に必要な就学年数＝〔0.39×(平均単語数/センテンス)＋11.8×(平均音節数/単語)〕−15.59

10.6　実証：フォグ指数で文章を編集する

初稿に対する編集の効果を明らかにするため、現在使用されている2つのマニュアルからテキストを引用し、実際に“編集前”と“編集後”の文章を示す。編集後の2つの文章は、いくつかの点で改善されているが、特筆すべきは、フォグ指数の測定値が示す通り、読む上での難解性が大きく緩和されていることである。

次に掲げる文章は、実際に使用されているマニュアルの一部であり、大手国際金融会社がプロジェクトを推進していく上で利用しているガイドラインである。

編集前：

Following identification of needs and appropriate preliminary approval of all major system development project proposals, the Information Systems Department will prepare an analysis and recommendation for action. The more routine requests will be approved by concurrence of the Information Systems Department and of the financial area management without further review. Those requiring a change in policy, exceeding the approved budgets or crossing organizational lines will require review and approval by the Steering Committee as well.

The Information Systems Department will evaluate the capability of the user or regional technical staff to implement a proposed system. Based on this evaluation, the responsibilities and authorities of the Information Systems Department, regional technical staff, and the user will be outlined in a system development proposal submitted to the Steering Committee.

単語：127、センテンス文：5、“難しい”単語：37、フォグ指数：21.6

少し手を加えれば、こうした必要以上に難しい（しかしよくあるタイプの）文章もいちだんと読みやすくなり、業務管理上の手続きも、プロジェクトの担当者にとって実行しやすいものとなる。

編集後：

First, needs are identified and major development proposals get preliminary approval. Then, the Information Systems Department analyzes each request and recommends as action.

For small, routine requests the Information Systems Department and the manager of the functional area may approve the project without further review. (A project is “routine” if it does not call for a change in policy, exceed current budgets, or cross organization lines.)

For major requests, though, the Steering Committee must also approve. To advise them, the Information System Department submits its own evaluation, which proposes schedules and tasks for all the participants.

単語：97、センテンス：6、“難しい”単語：12、フォグ指数：11.4

次の“編集前”の例として、FORTRANのプログラム・ガイドを掲げる。これは、大手のtime-sharing companyによって作成されたものである。

編集前：

It is critical that variables used as subscripts in FORTRAN programs always be consistent with information declared in the DIMENSION statements. Unless checking is specifically requested, subscript ranges are not checked for validity when programs are run. This checking is omitted in order to maximize running-time efficiency. However, if invalid values are used for subscript variables, such as a value less than one or greater than the maximum subscript as specified in the DIMENSION statement, errors can occur. Often such errors either go undetected or cause apparently unrelated failures and diagnostics.

When invoking the FORTRAN compiler, the user can inform the compiler that subscripts are to be checked for range validity by supplying the SUBCHK option.

単語：115、センテンス：6、“難しい”単語：21、フォグ指数：14.8

この“編集前”の文章は、２～３回読み直さないと理解できない。１回だけで理解できるプログラマーは、この内容について、すでに知っている者である。一方、“編集後”の文章は、特定の誰かを対象として“程度”を下げることもなく、一般の人々の大半に内容を正しく伝えている。一般の人々も以下に掲げる文章の方を選ぶであろう。

編集後：

Variables used as subscripts in FORTRAN programs must stay within the range of those in the DIMENSION statements. (That is, the value of the variable must not be less than 1 or greater than the highest subscript in the DIMENSION statement.) If they are out of the range, invalid, the mistake is often overlooked. Worse, these errors often cause “unrelated” failures or odd diagnostic messages.

To save running time, this system does not check the range of the variables unless told to. To be safe, when you invoke the FORTRAN compiler, tell it to validate the values with the SUBCHK option.

単語：100、センテンス：6、“難しい”単語：8、フォグ指数：9.8

10.7 マニュアルの吸引力を高めるその他の方法

読者を引き付けるマニュアルを作成するためには、マニュアル作成者は、読者の気をそらす原因をできる限り取り除き、マニュアルの中身を信頼できる形で提供し、加えて効果的なページのレイアウトを考えなければならない。

●読者の気をそらす原因

マニュアルおよび情報製品は、「読者の注意を引き付けるもの」でなければならない。読者の気を散らせる原因は、ほとんどがうっかりミスや製作上のミスである。誤字・脱字、おかしな位置の区読点、文法上のミス、用語の不適合、頭字語や略語の説明不足、へたくそなレイアウト、ぼけた写真や色分解、などがそうである。これらのバグは、その存在自体は他愛のないもののように思えるかもしれないが、インストラクションの効力を薄れさせ、読者の集中力を散漫にするには充分である。さらに、これらのバグが頻繁に登場し、バグがあることがあたりまえということになると、これはもう不注意でだらしがないと思われても仕方がない。つまり何のことはない、読者に最悪のメッセージを送っているのと同じことになるのである。

こうした意味からも、すべてを読者からできるだけ「信頼感を引き出せる」ような形で行なうことが重要である。小さなバグにも注意を払って編集作業を行なうのもいいだろう。印刷や製本において高品質を確保するのもいいだろう。マニュアルに高価な紙を使用すべきか否かを問うのは、もっとも難しい。しかし高価な紙を使えば、安手の紙を使用するよりも、ユーザーの反応がよくなることは間違いない。

●“バーゲン品”は高くつく

マニュアルの印刷がプリンターで行なわれるのなら－つまり人間ではなく機械でという意味だが－インクが裏まで“にじみ出ない”ように、充分厚い紙を使用する必要がある。機械は“バーゲン品”であっても、マニュアルの出来上がりが“安物”であってはならない。

安くつくと思えるもの、またそれ自体が安くみえるものは、すべてマニュアルの有用性を低下させる恐れがあることに注意して欲しい。これらはたいてい、マニュアルに対する読者の信頼感を損ねるものなのである。真剣に作られなかったマニュアルは、読者も真剣に読もうとはしない。したがって、安価な材料や方法でマニュアルを作成すると、多くの場合、読者をほとんど寄せ付けないものとなってしまう。たとえば、1ページにできるだけいっぱい文字を詰め込もうとする、大きな活字を使わない、太字を使わない、“レタリング”を入れない、その他、高くつく書体はいっさい使わない、といったような方法で作られたマニュアルは、読者にとって苦痛でしかない。

●紙の節約は努力の節約

さらに、マニュアル作成者は、紙の節約を第一義と考えるようなマネジャーには、警戒の念を抱かなければならない。**紙を節約しても、コミュニケーションが効率よく行なえるわけではない。**余白を大きく取り、大きな字で印刷する方が、読者にとっては都合がよい。それもすべての読者にとってである。きちんと整理されたページ構成は、読み疲れしにくく、したがって読み間違いも少なくなる。「厚い紙を使う」「装丁を丈夫にする」「章の切れ目にタブを入れる」「読みやすい書体や色を選ぶ」といったことは、本質的ではないにしろ、システムからその本来の有用性を引き出す助けとなるのである。

マニュアル作成者はまた、無理な言葉の省略、頻繁な略字の使用、その他の短縮形によってマニュアルのページ数を節約しようとするような札付きの編集者にも警戒しなくてはならな

い。同じ意味の言葉について、一方に明快で簡潔な書き方があり、もう一方に短縮されて意味の通りにくい書き方があるが、これらの間には大きな違いがある（この文章の中で、「一方に」と「もう一方に」という言葉を削除しようとする編集者は、私のいいたいポイントを理解していない）。

●編集の専門家を雇え

最後にいいたいことは、大量のマニュアルを作成している企業や組織は、有能な専門職の編集者を抱えるべきだということである。プログラマーも教育次第では、少しはましな文章が書けるかもしれない。ワープロは、文法上のミスやスペルのミスはチェックしてくれるし、フォグ指数の計算もやろうとすればできる。しかし、「内容をうまく伝えるためには、忍耐と繰返しが必要である」ということを理解している人、「簡潔性と密集性の違い、緊密性と乱雑性の違い」が分かる人が、必ず誰か必要である。そして、「マニュアルの作成作業の諸要素を“ランク付けする”ことがいかに恐ろしいか」を知っている人がたぶん必要となるだろう。

図表 10.7　＜用紙の節約　対　可読性（読みやすさ）の向上＞

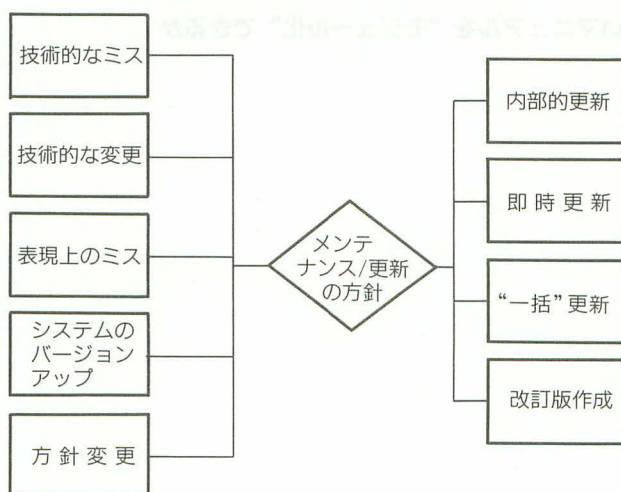
用紙を節約する場合	ユーザーの便宜を考える場合
狭い余白	広い余白
小さな活字、ぎっしり詰まったレイアウト	大きな活字、さまざまな書体サイズ
イラスト、図表を最低限に抑える	大きな図表、イラストを頻繁に使用
セクションごとの改ページがない	セクションごとに改ページ
繰返しの回避、最低限の説明	redundancy、読者を引き付ける工夫
タイプ打ちの表題や図表	正式な活字印刷による表題、本格的な図表

第11章

保守：マニュアルをサポート・更新する

11.1　マニュアルを保守する：刺激と反応

どんなマニュアルも、完成後の変更が必要になるだろう。事実、どの企業を見ても、ごく最近作られたものを含め、あらゆる文書に変更の必要性があるというのが世の習いではないだろうか。この変更という作業段階で、もっとも考慮すべき重要な課題は、文書およびその補遺を供給する際に、いかにうまく全体を管理するかということである。マニュアルの変更や改訂の衝動を1つの刺激とみなし、それに正しい反応を返せるようつかさどるルールをメンテナンス基準、あるいはメンテナンス・ポリシーと考えよう。

プログラムやシステムと同様、マニュアルなどのまとまった文書を確実に保守（メンテナンス）するにあたって、まず最初にやらなければならないことは、**保守の職務にあたる人を選ぶことである。**誰かが責任を持たなければ、おそらくその職務は遂行されないだろう。どの書類についても、担当者が誰で、それが正しく供給されているか否か、またその書類の更新や補足が適切な時期になされ、適切な人に送られているか否か、ということを管理しなくてはならない。

図表 11.1　＜刺激と反応＞

●メンテナンスを促す刺激

あらゆるマニュアルは、変更が必要になるだろう。図表11.1に示すように、いかに注意深く見直しが行なわれても、マニュアルは次のような刺激に応えなければならない。

- **技術的なミス**－システムやソフトに関する不正確な、あるいは不完全な技術的情報。
- **技術的な変更**－マニュアルの作成中にシステムに加えられるごく小さな修正。マニュアル作成者に知らされる場合もあるし、知らされない場合もある。
- **表現上のミス**－曖昧な、不明確な、あるいは誤解を招くようなテキストや図表、文法上のまたは誤字・脱字などのケアレスミスも含む。
- **システムのバージョンアップ**－ソフトやシステムになされる大きな変更または“新機能の

追加”。予定されている場合もあるし、そうでない特別な場合もある。
・**方針変更**－何をなすべきか、何をなし得るか、担当を誰にするかに関する新しいルール。

繰り返していうが、誰かが責任を持って、これらの問題を見失わないようにして、必要な努力を怠らないようにしなければならない。しかし、誤解してもらっては困るが、いくら責任があるとはいえ、つねに新情報の報告、通達、発表を際限なく繰り返す必要はない。

いずれにしろ、忘れてはならないもっとも重要なことがある。つまり、マニュアル作成者が、マニュアルを色のあせない、間違いのないものにしたいと思う気持ちはよく分かるが、**どんなマニュアルも、どのみち時の流れには勝てない**ということである。「マニュアルをいかに手っ取り早く更新あるいは訂正できるか」ということが問題なのではない。問題は「どの変更をしばらく保留にしておけるか－つまり“一括更新にできるか”－そしてどれを直ちに伝えなくてはならないか。一括にしたもののうち、どれを2、3日、あるいは2、3週間、あるいは数ヶ月保留できるのか」ということである。

●刺激に対する４つの反応

実際には、刺激に対して、次の4通りの応え方がある。

・**内部的更新**
マニュアルのマスターバージョンを修正することである。つまり、保守責任者のファイルに保管されている内容を変更することである。このファイルには、マニュアルの修正済み個所、修正の必要な個所、および出版スケジュールが書き込まれている。この内部ファイルにあるものは、すべて最優先事項であり、できる限り最新の状態が維持されている必要がある。

・**即時更新**
ユーザーやマニュアルの所有者全員に緊急報告を送ること。当然これは重要な伝達事項のために取っておかなくてはならない方策である。というのは、突然緊急報告が送られると混乱をきたし、またシステムが混沌とした状態に陥ったか、あるいは“狼が来た！”の類だろう、という印象を与えてしまうからである。

・**“一括”更新**
いくつかの変更事項をひとまとめにしたもの。定期的（月１度、3ヶ月に１度）、あるいは変更の量がある基準線を超えた場合に出される。

・**改訂版作成**
究極の更新である。この場合でも、定期的にまたは変更の量に従って、マニュアル作成者は、前回の編集以後の修正を、すべて新しいマニュアルに組み込む。つまり古くなったものを取り除き、修正されたものに置き換える。こうしてユーザーやDPセンターは、改訂版を受け取る。（古いものを返してから改訂版を受け取るのが理想である。）

つまり、反応の仕方はいく通りかあり、緊急の度合いもいく通りかある（あらゆる工学技術の問題がそうであるように）。それでも、熱心なマニュアル作成者は、その熱意のあまり、ひっきりなしに改訂を行ない、明確さよりむしろ混乱を招いてしまうこともよくある。

11.2　見越しの設計でメンテナンスを改善する

マニュアルその他の文書をメンテナンスしやすく、修正しやすいものにするためには、初めからこれらのことを考慮して設計しなくてはならない。モジュール化されたマニュアルで、しかもモデルの段階のうちにテストが行なわれているものであるならば、その後の変更の必要性が少なくなるばかりでなく、何らかの変更が必要である場合でも、その変更を実行したり管理したりするプロセスは単純なものですむ。

●メンテナンスは設計段階から

ソフトウェア・エンジニアリングの世界ではあたりまえのことになっているが、プログラムはその設計の初期の段階で、すでにメンテナンスの便宜を考えて作成される。プログラムが作成された後でもメンテナンスはできるが、設計段階で考慮された場合ほどたやすくはいかない。システムを“後で組み直す”試みは、見上げたことではあるが、むしろ無謀というべきだろう。こうした試みは、なるべく初期の段階でなされるに限る。出来上がった後の組直しがうまくいったためしはほとんどない。

メンテナンスのしやすさとは、システム内のバグや欠陥や不適当な部分をいかに簡単に、そしていかに手早く探し出し、明確にし、訂正できるかという総合的な目安である。システムがメンテナンスしやすいかどうかは、たいていの場合、システムのコスト効果を占う唯一最大の指標である。

これと同様に、コンピュータ・システムに付随するマニュアルやその他の情報製品も、やはりメンテナンスされ修正されるべきものである。システムに変更が加わったり、バグが発生したりすれば、それにともなってマニュアルも変更されなくてはならない。あるいは、マニュアル自体が、システムとは関係ないバグや弱点を現わす場合もある。遅かれ早かれ、マニュアルをメンテナンスしたり修正したりすることは、マニュアルを書くことよりも高くつくということが分かるはずだ。そして、メンテナンスしたくてもできないマニュアルは、おそらく失敗作ということになるだろうし、そのマニュアルによってサポートすべきシステムの使いやすさを大幅に引き下げる結果にもなる、ということも分かるだろう。

●モジュールの登録簿を作る

マニュアルをメンテナンスするもっとも簡単な方法は、第一に変更する必要のあるモジュールを見つけ出すこと、次にそれを訂正するか、あるいは別のものと置き換えることである。マニュアルを修正する場合でも、どのみち訂正や置換えの代わりに、モジュールを付け加えるべき正しい位置を見つけ出し、それをそこに挿入するという作業をともなうこととなる。

マニュアルがモジュール化され、構造化されていれば、すべてのモジュールの登録簿を保守（メンテナンス）することができる。その登録簿では、各モジュールに関し、該当するシステム、項目、適応業務、導入場所、その他関連する詳細な記述子などが、コード番号などで明記される。このようにモジュールを登録簿で管理しておけば、システムに何らかの変更があった場合、それに影響を受けるすべてのモジュールを探し出すことができる。また、古いモジュールから新しい文書を作り出すことも可能なはずだ。

マニュアルが、モジュールの組合わせによってのみ構成されるものであると考えるなら、登録簿は図表11.2aのように保守することができる。1つのモジュールが数ヶ所で用いられることは充分考えられるので、このような登録簿によって、システムの変更に影響される「すべて」のマニュアルの部分を知ることができる。

図表 11.2a ＜モジュールの登録簿から＞

モジュール名：レコードの追加　　**モジュールファイル番号**：B-008
最初の30字：
　レコードを追加するには、ファイル名を GOTO win に入力する
上位モジュール：B-008　4つのファイル処理を利用する
下位モジュール：B-028　既存のレコードを追加する
　　　　　　　　　B-029　キーデータなしでレコードを追加する
詳細
　プログラム/システム：DB-3 不動産管理システム、貸付管理システム
　適応業務：ファイル作成、ファイル更新、新規勘定、新規レコード
　対象：エンドユーザー、不動産仲介業者、貸付管理者
　導入先：ABCO ファイナンス、Goldschmidt & Wong 不動産
　その他：

当該モジュールを含むマニュアル/情報製品：
　G-3、G-4、G-5、F-1、F-2、F-7、R-1、R-5

図表 11.2b ＜出版したモジュールの登録簿＞

モジュール番号	等価モジュール番号	マニュアル/情報製品番号
B-008	R-006、D-120	G-3、G-4、G-5、F-1、F-2、F-7、R-1、R-5
B-028	R-061、D-121	G-3、G-4、G-5、F-7、R-5
B-029	R-062、D-122	G-3、G-4、G-5、F-7
C-110		G-2、G-3、G-4、G-5
C-115	R-090	G-4、G-5、R-5
C-240	D-600	G-3、G-4、G-5、F-7、R-4、R-5
・ ・ ・	・ ・ ・	・ ・ ・

●メンテナンスを見越してマニュアルをデザインする

大規模な先進企業では、“等価モジュール”といわれるような、同じ技術的内容を持つ、交換可能なバージョンを取り入れているところもある。図表11.2b に示す登録簿を見れば、マニュアル作成者は、技術的な変更が行なわれた結果、どのマニュアルを変更しなければならないかを一目で見ることができる。もちろん、この図はどちらかといえば複雑で大がかりである。もっと簡単な見越しの設計を選べば（つまり、マニュアルを前もってメンテナンスしやすい設計にすれば）、それによってマニュアルはもっとメンテナンスしやすくなる。たとえば、ルーズリーフのバインダーに収められたマニュアルは、当然製本されたものよりも変更しやすい。（しかし逆に、ある種の間違った情報は、あえてルーズリーフを用いないことによって、未然に防ぐことができる場合もある。）

1ページずつの、つまり用紙の片面にだけ印刷したモジュールは、もっとも簡単に追加や削除や挿入ができる。したがって、絶えず変更の必要があるマニュアル用のモジュールとしては、おそらく最良の形式であろう。しかし一方、1ページモジュールにすると、一貫性のない、複雑な参照や繰返しの多いマニュアルになりやすい。1ページモジュールー本文と図表が同じページにあるものーはメンテナンスという点では優れているとしても、2ページモジュールの長所である信頼性と可読性（文書の読みやすさ）には代えがたいものである。

11.3　メンテナンスのパラドクス：やりすぎは逆効果

マニュアルのメンテナンスは、よほど注意深い方針を立てて行なわないと、補遺、速報、リリース版、最新版などの配給がいい加減になったり、でたらめになったりしてしまう。多くの人が気付いていないことだが、マニュアルに付加情報を加えるたびに、違うバージョンのマニュアルが、実際世にあふれてしまうのである。そして、これらのバージョンの中でも、正しいのはたった1つしかない、ということになってしまう。

●情報伝達の熱力学的法則

またしてもあたりまえのことだが、時の流れに逆らえるマニュアルはなく、マニュアルには間違いが付きものである。もっと明確な、あるいは絶対に否定し得ないという言い方をすれば、複雑なプログラムやデバイスには、複数のバグが－確認されていないものも含めて－例外なく存在しているのである。

もう1つあたりまえのことをいおう。ユーザー・リスト、配布先リスト、配布経路リストにも例外なく誤りがある。そして、リストが長くなればなるほど、ずさんなものになり、時の流れに対応できなくなる。つまり、あるマニュアルの読者（あるいはシステムのユーザー）に、変更やその要点を伝えようと思っても、上記のリストがずさんで不正確であれば、その試みは挫折するのである。もちろん例外もある。プログラマー、オペレーター、ユーザーの3者が同一人物の場合である。しかし、このような場合、どのみち正式な基準にのっとったマニュアルであることはまれである。

上記の2つのあたりまえのこと、つまりどんなマニュアルも完璧ではなく、どんな配布先リストも完璧ではないということは、もはや技術的な情報伝達における自然法則である。これに、熱力学の第二法則*（エントロピーの法則）を加えるならば、なぜマニュアルの更新と修正にあまりにも多くの挫折があるのか理解できよう。

どんな変更が必要かを知り、その報告をまとめるという不断の努力も必要である。補遺や最新版の配給対象となる人々と、その宛先を確認するという気の遠くなるような努力も必要である。しかし、恐ろしいのは、何か得体の知れない手に負えない巨大な力が働いていて、そうした努力をくじき、ねじ曲げようとしていることである。郵送システムには、官民を問わず、間違いが付きものである。たとえ電子メールやファクシミリを使っても、誤りはなくならない。また、受取人が配布された付加情報を、誤った場所に置く、誤った用途に使う、誤った解釈をする、さもなくば濫用するといったこともある。たとえば、付加情報が必要であるにもかかわらず、まだ追加されていないマニュアルが、いったいどれだけ読者の手もとに眠っているのだろうか。

●バージョンアップは旧バージョンを増やす

マニュアルに情報を追加することは－たとえそれが時流に対しても信頼性に対しても色あせないマニュアルを作ることが目的でなされたにせよ－違うバージョンのマニュアルを、世に氾濫させる結果になりかねない（つまり、入手した人とそうでない人の2種類の読者を産む恐れがある）。マニュアルのオリジナルが発行されたときは、1種類のバージョンしかない（もちろん、勤勉なユーザーによって抄出され、まとめられた非公式のバージョンは勘定外である）。しかし、マニュアルに情報が追加されるたびに、マニュアルのバージョンは、ねずみ算式に増えていくことになる。

つまり、補遺を2回発行すると、使用されるバージョンは4種類になり、4回発行すると16種類

になる。10回も発行すれば1キロバージョン（1024）にものぼる！

話をおもしろくしようとして、こんな説明をしているわけではない。かなりの人数のオペレーターやユーザーに対して、訂正版や更新版を送ったことのある人なら分かるだろうが、使い方のミス、設置場所のミス、管理のミスなど、起こり得ることは、事実起きているのである。

差し出す側が、間違った内容を提出することがなくなったとしても、受け取る側が、それを間違いなく運用するとは限らない。旧バージョンに取って代わるべきものが、机の中にちゃんと入っているのに、旧バージョンが、相変わらず用いられている場合もある。

●情報の氾濫を防ぐ４つの方法

誤った情報のたれ流しを完全にくい止める方法はないが、以下のような事柄を実施すれば、事態の改善につながる。

・補遺の発行回数を制限し、それらをひとまとめにして蓄えておくと、マニュアルの配布経路に生じるノイズを低減できる。また、正規のスケジュールにしたがって、その補遺のまとまりを放出すると、もう1つの切実な問題を解決するのにも役立つ。つまり、必要な追加情報は「すべて」受け取っている、という信頼感を読者に与えることができる。マニュアルの更新が不定期に行なわれるなら、ユーザーが受け取るべき情報に漏れはないという確信を持つことはあり得ないだろう。

・マニュアルに書かれるべき内容を、できるだけ多くシステムそれ自身の中に組み入れてしまう。これによって、“印刷物”というどんどん古くなっていくものの量を減らし、問題の発生を抑えることができる。

・更新を行なう際、情報源を単一の公認されたものに限定するのも効果的だろう。

・補遺が発送される前に、書かれた内容を技術面の専門家に点検してもらい、認可してもらう。これによって、更新や訂正の回数を減らすことができる。特に、訂正の訂正という事態を避けることができるだろう。

訳注＊熱力学の第二法則：「エネルギーは秩序から無秩序へ、使用可能なものから使用不可能なものへという、1つの方向にのみ不可逆的に変化する」。たとえば、角砂糖をコーヒーに入れると溶けるが、溶けた砂糖はコーヒーと分離して再び角砂糖には戻らない（不可逆性）。そして溶けた砂糖は、角砂糖のときよりもエントロピーが増大している。つまり、エネルギーは、つねにエントロピーを増大させる方向にのみ変化する。本書では、いったん流通経路に乗せたマニュアルを、すべて回収して誤りを訂正することはできない、という意味と思われる。

11.4　古いマニュアルを“モジュール化”できるか

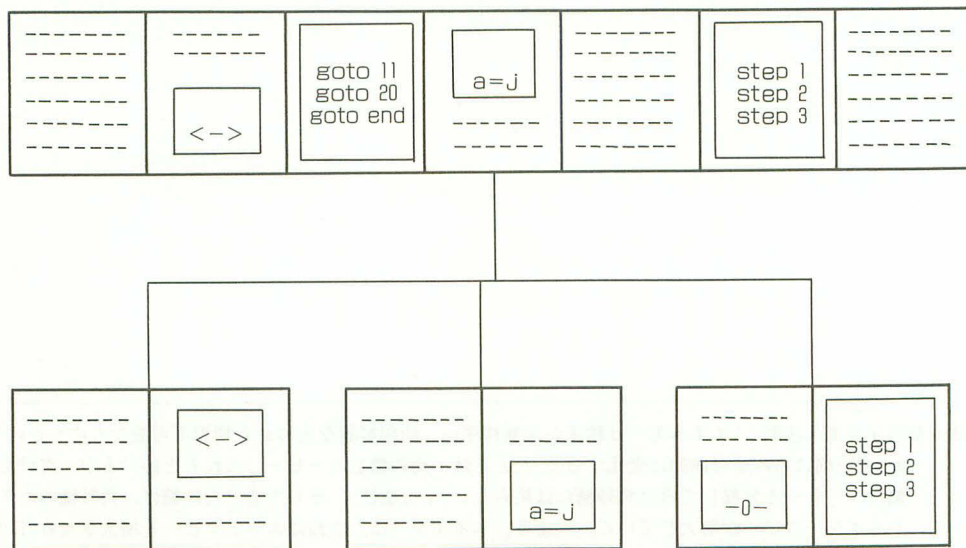
ゼロから始まるマニュアルはほとんどない。古いマニュアルを混ぜ合わせるか、更新するのが普通である。まれに古いマニュアルが“構造化された形式”に書き直される場合もあるが、このようなやり方は、うわべだけの整形手術になりがちだ。新規に作られてテストされたモジュール化マニュアルの利点を、本当に備えているものではない。

●事後のモジュール化は可能か

新規採用されたマニュアル作成者の任務の多くは、あまり評判のよくないマニュアルを手直しして仕上げる、更新する、グレードアップする、改訂する、さもなくば“整理整頓する”というものである。彼らが仕事を始めるように頼まれるのは、前章までに説明してきたようなマニュアル設計に関する決定が、ほとんどなされてしまった後なのだ。

既存のマニュアルに関してはどうか。古いマニュアルを編集、あるいは改訂する仕事を請け負わされた作成者は、構造化の方法を利用することができるだろうか。手の付けられない、信頼性の低いマニュアルが、それ以上使いやすくなるだろうか。

おそらくそれは可能である。Hughes Aircraftの編集者たちは、初めてモジュール化の方法（STOP*テクニック）を公表したとき、古い文書を新しい2ページのフォーマット（形式）に作り変えることができた、と報告した。彼らの成功は、ある程度手掛けた文書の性格に負うところが大だった。つまりそれらの文書の多くは、すでに本文と図表が、均等にミックスされたものだったからである。

うってつけのマニュアルが与えられれば、仕事のプロセスは、ほとんど遊びのようなものだ。まず、既存のマニュアルを1ページずつ切り離し、順番に並べて置く。そして、本文と図を吟味し、内容をモジュールのサイズに区切って印を付け、新しいヘディングあるいはヘッドラインを書き込み、必要ならごくまれだが、節や段落を組み換えたり、図を付録部分から本文に移動

図表 11.4　＜旧マニュアルを組み直す＞

させたりする。

●モジュール化はうわべの整形手術ではない

この考えがよいかどうかの判断を下すには、いくつかの要因を検討しなくてはならない。すでに述べたように、あるマニュアルが、他よりも構造化されたフォーマットに近ければ、それだけ複雑な改訂作業を必要としなくなる。本文や図がよく書けているマニュアルは、残す（再利用する）だけの価値があり、その努力は報われる。しかし、マニュアルの多くは（古いプログラムと同様)、すでに時代遅れで役に立たなくなっている。これらを残そうとすることは、「新しいものを作り出すより、既存のものを再利用するほうが、時間と金の節約になる」というあまりに通俗的な神話を認めてしまうにすぎない。

繰り返すが、古いマニュアルの中には、ごく簡単に“モジュール化”できるものもある。そもそも最初からストーリーボードなどなかった既存のマニュアルから、ストーリーボードを“ひねり出した”つわものさえいる。しかし多くの場合、**古いマニュアルを作り直したり、組み直したりする方が、新しいものを作り出すより、もっとしんどい作業である。**

古いマニュアルの物理的な構造やフォーマットを、より読みやすいモジュール化されたフォーマットに作り変えることが可能な場合でも、次のことは覚えておいて欲しい。モジュール化の方法は、マニュアルの内容に人目を引き付けるためだけの方法ではない。それは、ただうわべだけの整形手術ではないのだ。

モジュール化によるマニュアル設計は、その美顔術的意味合いはともかく、マニュアルのメンテナンスのしやすさや、修正のしやすさを確実にするための道でもある。すべて出来上がってしまった後（事後）にフォーマットを強制的にモジュール化しても、外見を良くすることにはなっても、校訂されてしまったマニュアルのメンテナンスのしやすさや信頼性は、たいして改善されはしないだろう。

Yourdon と Constantine は、古いコンピュータ・プログラムの作り変えに関する討論で次のように述べているが、私は彼らの意見に賛成したい。

既存のプログラムやシステムの構造を、事後のモジュール化を通じて大幅に単純化するのは、ほとんど不可能である。ひとたびコーディング（プログラミング）が行なわれてしまうと、システムの構造の複雑さは本質的に定着してしまう。したがって明らかなのは、単純な構造にしたければ、最初からそれなりの方法で設計しなくてはならないということだ。－*Structured design*（Englewood Cliffs，NJ: Prentice-Hall)，1979，p.35

その気があるなら、古いマニュアルを作り変え、組み直してみるとよい。しかし、最初からマニュアル作成を成功させるチャンスを逸してはならない。最初からモジュール化の方法でマニュアルを作るのと、そうでないのとでは有用性とコスト効果において、顕著な差が出てくる。

原注＊ STOP：Sequential Thematic Organization of Publications の略。1960年代初頭に Hughes Aircraft corporation が開発した文書作成技法。

マニュアルの将来

第12章

本なきマニュアル

12.1　マニュアル不要のシステムは可能か

確証はないが、印刷物としてのマニュアルは、そのページ数が減っていく傾向にあるようだ。ユーザー向けの情報は、供給され利用される機会がどんどん増えているが、それは印刷物という形を取っていない。もっと重要なことは、オンライン・システムの特性が活かされていることである。オンライン・システムは、年々改良が施されるにつれ、それ以外の付随的なユーザーサポートをあまり必要としないようになってきている。

●マニュアルは隅に追いやられる

コンピュータ技術を利用している人々の大多数は書物を好まず、またそれをうまく活用しようともしない。こうした人々の数はさらに増え続けている。これらの人々の一方の極に位置するのは、事務職員やデータ入力要員の一団で、彼らは能力もあり、才智に富んでいるが、分かりにくいマニュアルの中から自分の探しているものを見つけ出す術を知らない。またもう一方の極には、重役や管理職たちが位置する。彼らはあまりに多忙で、あるいは短気であるため、3節以上の文章を読む気はなく、誰かに質問してその答えを待つことに慣れてしまっている。

マニュアル（特に百科事典的なもの）は、邪魔な存在にもなり得る。驚くべきことに、ほんの2、3の部署しか本棚を備えておらず、マニュアルを開いて読める場所のある部署はもっと少ない。そこで、2冊のマニュアルを膝の上にバランスよく置いてシステムに向かう、というような人間工学を無視した事態が頻繁に発生することとなる。

印刷物としてのその他の情報製品もまた、時流に取り残されずにいることの困難なものにくみする。“ハードコピー”（これはふさわしいネーミングだが）は、ディスクやテープに収められた形で供給される情報などと違い、それほど容易に変更のきくしろものではない。したがって、印刷物としてのマニュアルは古くさく、時代遅れとなってもなおしぶとく存在して、使用され続ける可能性がある。

従来のマニュアルにも、まだまだいくつかの利点がある―とりわけ、われわれのようにそれらを再チェックしたり、編集したりする者にとっては―とはいうものの、ある種のシステムを導入する場合には、出版物としてのマニュアルを追い払う作業が当然必要となってくる。システムがマニュアルをまったく必要としなくなることはないが、資料をほとんど必要としないシステムはあり得るだろう。

●マニュアルをなくす方法：その1

出版物としてのマニュアルを排除するという目的を達成するためには、2つの一般的な戦略がある。難しい方の戦略からいうと（結局はより効果的なのだが）、ユーザーが最初のオリエンテーションを受ければ、後はほとんど説明を受けずにすむようにシステムを設計することであり、またオペレーターやユーザーがいっさい何も調べずにシステムを操作できるようにすることだ。この第一の作戦のキーポイントは、画面（特にプロンプト画面やメッセージ画面）を、平易な言葉でコミュニケーションが図れるよう工夫することだ。「**INSERT FMT DSK**」と表示するよりも「**フォーマット・ディスクをインサートしてください**」と表示した方が、マニュアルの必要性は減少する。また、「**チャートを作成する前に、フォーマットしたディスクを挿入してください。**」と表示した方が、マニュアルの必要性はさらに低くなる。（選択したチャートには、フォーマットしたディスクのうちのどれがいいのかを示してくれる画面の方がもっとよい。ひとたび作業を開始したら、ユーザーにどんなディスクも選択したり、見つけたり、インサートすることを要求しないシステムが一層よい。）さらにいえば、ユーザーが、フォーマット・ディ

スクの種類をリストアップした表を参照しなければならないなら、その表を画面上に表示させることで、一層マニュアルの必要性は減少する。

なぜ何かを参照するという行為が必要かといえば、それは一般的にメッセージがコード化されたり、暗号化されたりしているからである。たとえば、「**エラー17**」というようなメッセージは、「**ライトエラーの項目参照**」というメッセージよりサポートを必要とするが、後者もまた下記のメッセージより多くのサポートを必要とする。

このファイル内のレコードは変更できません。
まずファイルの最初のレコードを確認してください。

もちろん、もっとも効果的な方法は、システムをエラーが発生しても自動的に復帰できるようなものにするか、エラーそのものが起こらないようにすることである。

エラーへの耐容力が強いシステムほど、マニュアルの必要性は低くなる。画面表示が乱雑でなく、略語や短縮語に頼らないシステムも同様に効果的だ。結局、システムを改良する最善の方法は、**大量の説明を要する手順を、説明不要となるまで修正する**ことだ。

●マニュアルをなくす法：その2

マニュアルの量を減らすもう1つの戦略は、言葉や図表を電気信号に変えることである。現今では、それはたいていシステム自身のディスプレイに表示されるものを意味する。表示される内容には、典型的なものとして2つのケースがある。1つは対話式教育用プログラム（通常ソフトウェア産業でいう“チュートリアル”）である。もう1つは「ヘルプ」機能（ユーザーが必要となったときに、指導的なあるいは参照的な内容を表示する一連の画面）である。

このチュートリアルやヘルプ画面には、何ら新しい内容は含まれていないことに注意して欲しい。チュートリアルとは、プログラミングされた教科書のようなものであり、ヘルプ機能とは、内容検索のためのプログラムが付いた一種の参考書である。それらは決して目新しいものではないが、多くの場合、必要な対処策を行なうためのよりよい方法である。事実、チュートリアルやヘルプが適切にプログラミングされているなら、それらは本書に述べられているすべてのマニュアルの長所を備えているはずであり、マニュアルを利用しづらくしている回り道や枝分かれを完全に削除しているに違いない。換言すれば、うまく設計されたオンライン・マニュアルは、文書化されたマニュアルをほとんど不必要なものにすることができる。

文書化されたマニュアルにせざるを得ないものは、導入の手引とスタートアップ・ガイドである。すべてのマニュアルがシステム「内」にあるなら、システムの「外」にあるマニュアルのみが、システムの起動方法を教えてくれることになる。同様にいえば、プロンプト表示は、次の操作を指導するものであり、ヘルプ画面は、問題を解決するのに役立つものである。

しかし、情報製品と情報サービスとの間に交換可能な関係があるように、文書化されたマニュアルとオンライン化された情報製品の間にも交換可能な関係が成立する。そのどちらにも「本来的な」優位性はない。正しい分析、設計、テストが行なわれないと、オンライン・マニュアルはもっとも複雑なマニュアルと同様、使いにくいものとなる。

12.2 ユーザー・インフォメーションをオンライン化する

マニュアルに印刷されている内容はたいていの場合オンライン化できるが、はっきりさせるべきことは、単に端末のディスプレイにマニュアルの内容を載せ換えれば、使いやすくなるわけではないということだ。

●画面は一度に1つしか見られない

マニュアルをオンライン化する上で忘れてはならないもっとも重要なことは、このマニュアルのオンライン化という方法には、特に秘法があるわけではないということだ。パソコンが出現するまでは、文字を映像化するもっとも一般的な方法は、「マイクロフィルム」であった。しかし、マイクロフィルムを使わなければならない学生、研究者、あるいは資材部の責任者などのほとんど全員が、このシステムを不便で気乗りのしないものだと感じていることに注意して欲しい。

新しい技術にはありがちなことだが、マニュアルをオンライン化する主な利点は、またその欠点でもある。従来のディスプレイ画面は、25×80文字というコンパクトなものであるため、ライターは情報を簡潔にまとめ、極小のモジュールに詰め込まなければならない。ユーザーが一度にただ1つの画面しか見られないという事実がある以上（“ウィンドウ”については次に述べる)、マニュアル作成者は、そこに表示させる内容をうまくまとめざるを得ない。よって、オンラインによるユーザーへの指示は、初心者、つまり限られた読解力しか持たない読者にとって理想的なものとなる。

しかし一方では、画面がコンパクトであるがために、充分な説明や実演をするには、オンライン・モジュールでは、あまりにも小さすぎるという事態が起こってくる。2ページを一度に読むことが難しいのと同じように、2“ページ”分のオンライン画面を一度に読むこともほとんど不可能に近い。“ウィンドウ(画面上にポンと出てくる長方形の窓で、その中にテキストが表示される)”を使ったとしても、たいした助けにはならない。1つのウィンドウ中に収まるものが、1つの画面の中に収まるものより多いということは決してない。それでは、1つの画面の中に3つか4つのウィンドウを開いたらどうか。これは、コンピュータ・ショウでのデモンストレーションをおもしろくするかもしれないが、たいてい分かりやすくなるよりも、混乱を引き起こすことの方が多い。

もちろん、これらの難点は、決して解消できないものではない。50列以上の文字表示ができる“フルスクリーン”ビデオ・ディスプレイがすでに存在するし、また1つの画面上に複数のフルスクリーンを同時に映し出すことのできるウィンドウ・ソフトウェアもある。最終的にはこうした技術がかなり一般的に普及するであろう。

それまでは、オンライン・スクリーンが普通の本のページより小さくなるのは仕方のないことだろう。しかし将来、2ページ分の広がりで、3倍の量を乱雑にならずに詰め込むぐらいのことは可能になるだろう。とはいえ、ある1つの概念なり操作手順なりを一目で見ることができる、いわばテキストと図表のすべてを1ヶ所に収めることができるという点を過大視するならば、オンライン・マニュアルはその適応範囲を限定してしまうことになるだろう。

```
MANUAL12.JXW　非書式 28 行 33 桁　　　挿入　標準
編集メニュー
^Blocks ブロック編集　^Print 印字制御　^Locate 検索／置換　^Get Help
^Cursor カーソル移動　^Tabs タブ設定など　^Next 次候補　説明表示
^Option オプション　^Key 短文登録　^Remove 消去　^Quit
　　　　　　　　　　　　　　　　　^Undo 消去の復活　終了
* f･9 キーで漢字／英語入力モードを選択
T1 T 2 T 3 T 4T T5 T 6 T 7 T 8
１２．２　ユーザー・インフォメーションをオンライン化する
　マニュアルに印刷されている内容はたいていの場合オンライン化できるが、はっきり
べきことは、単に端末のディスプレイにマニュアルの内容を載せ換えれば、使いやすく
けではないということだ。
●画面は一度に１つしか見られない
　マニュアルをオンライン化する上で忘れてはならないもっとも重要なことは、このマ
ルのオンライン化という方法には、特に秘法があるわけではないということだ。パソコ
現するまでは、文字を映像化するもっとも一般的な方法は、「マイクロフィルム」であ
しかし、マイクロフィルムを使わなければならない学生、研究者、あるいは資材部の責
どのほとんど全員が、このシステムを不便で気乗りのしないものだと感じていることに
て欲しい。
B編集 C移動 O選択 P制御 T設定 R消去 L検索 N次 漢字 MORE1
```

図表 12.2　オンライン・インフォメーション（画面のフォトコピー）

●オンライン・マニュアルにも設計が必要

ほとんどの場合、オンライン・マニュアルは、次に何をしたらよいかを初心者に伝えるためのものである。それゆえに、マイクロプロ社のワードスター®のような洗練された製品では、ユーザーが画面上の参照情報を一部分に限定したり、それを画面からすべて締め出したりすることができるようになっている。図表12.2が示すように、もしもユーザーがワードスター®の“ヘルプ”を最大限表示させせようとしたら、画面が参照項目で埋めつくされ、文書はほとんど見えなくなるだろう。この製品のユーザーのほとんどがコマンド用の参照カードを利用する理由はここにある。

もっと性能のよいビデオスクリーンや新しいソフトウェアが登場すれば、これらすべての不平不満が、近視眼的なものだったということが分かるかもしれない。大型でもっと読みやすいディスプレイは、不自由な状況を変えてくれるに違いない。その上、「サブ画面」がマニュアルのために使えるのなら大助かりである。特に、サブ画面が、画像や写真を鮮明に映し出すことのできる高解像度ディスプレイ装置であるなら文句はない。

しかしながら、技術がどれだけ進歩しても、マニュアル作成に関する課題は相変わらず残るだろう。なぜなら、スマートなタイプライターでくだらない文をタイプすることが可能なように、ヘルプメッセージに有害な情報を書き込むことも可能だからである。オンライン・マニュアルは単なる「媒体」でしかなく、ユーザーが要求し、必要とするものを伝達する方法でしかない。オンライン・マニュアルは、従来のマニュアル同様－いやそれ以上に－注意深く企画・設計されなければならない。

第13章

あとがき：第五世代におけるマニュアル

13.　あとがき：第五世代におけるマニュアル

次世代のハードウェア（卓越した記憶機構とスピードを兼ね備えている）やソフトウェア（卓越した“知能”を持つ）は、従来のマニュアルの必要性を軽減するであろう。そしてゆくゆくは、現在マニュアル作成に費やされている技術が他の方向に向けられるだろう。つまり、主にアプリケーションを開発したり、コンピュータと人間とのコミュニケーションをより充実したものにするという方向にである。同時に、この新しいテクノロジーは、並外れて優れたマニュアルを「作成する」という方向にも使用されるようになるだろう。

●第五世代はコミュニケーターが開発リーダー

システムが使いやすいほど、そのマニュアルは書きやすい。システムの設計とそれを実現する技術（エンジニアリング）がすばらしければ、マニュアルにとっては、それだけで充分である。開発者がシステムに新しい特徴や機能（財務管理システムでのカラー選択、ステレオのイコライザー、電話のためのメモリーなど）を付け加えた場合のみ、ライターは創作意欲を呼び起こされる。

このことは、大学がプロのマニュアル作成者やテクニカルライターの養成を始めてわずか2、3年しか経っていないのに、彼らの存在がすでに時代遅れになりかけている、ということを意味するのだろうか。もちろんそうではない。この新しい職業が、そんなに早く姿を消すことはない。

そもそも、マニュアルライターという職業が衰退しても、そう簡単に彼らの仕事がなくなることはないだろう。

しかしながら、コンピュータ・テクノロジーの最先端では、従来のマニュアルは間もなくあまり必要とされなくなるだろう。システムはユーザーにとってますます“意識しなくてよいもの”となるため、“人間との接点”は、ほとんど見られなくなるだろう。こうした状況下では、システムとユーザーが意思の疎通を図る場合、コンピュータのプロンプトは分かりやすいし、コンピュータ自身がユーザーの意思を推し測る能力を持っていることになるので、ユーザーへの“サポート”の必要性は、ますます軽減されるだろう。

“サポート”、すなわちうわべはユーザーがシステムから利益を得ることを助けるという意味だが、実はむしろユーザーの方を知らず知らずのうちにシステムに適応させようとする念の入った表現であることの方が多い。システムの方がユーザーに適応するようになったら、ライターは何をしたらよいのだろうか。

それは明白だ。彼らはシステムの人間的かつ知的な部分を設計すればよい。それは、プロンプト表示やオンラインの参照画面を書く要領でできる。**第五世代およびそれ以後も、プロのコミュニケーターの技能は、エレクトロニクスやソフトウェアのエンジニアの技能よりも、システムの成功にもっと欠かせないものとなるだろう。**それればかりか、テクノロジーとその可能性についてもよく分かっているそうした有能なライターが、技術者の一団が通り過ぎた後の混乱を整理するという役に甘んじず、開発のリーダーになるかもしれない。

●これからは“書き手パワー”の時代

また、第五世代のテクノロジーが、ライターやマニュアル作成者自身の上におよぼす効果について考えてみるのもおもしろい。5年以上に渡って、私はクライアントに、どんな組織においても、書く作業をする人は直接ワープロを使って仕事をすべきだと提唱してきている。それは、何もタイピスト代を倹約するためではなく、それぞれのライターの表現力を高めるためである。

ペンやタイプライターからワープロに切り換えたほとんどの人が経験すること、特にプロのライターにとって、それはワープロによって自分の仕事の質が高められるということだ。ワープロはあらゆる編集・校正作業を簡単にしてくれ、そのため、編集・校正作業はより活性化される。すなわちそれは、頭に浮かぶ言葉と現実の紙との間の隔たりを縮めてくれ、カット・アンド・ペースト（切貼り作業）を思うがままにさせてくれる。そればかりではない。通信機能と結び合わせれば、こうした新しいテクノロジーは、マニュアル作成のプロジェクトチームに“ネットワーク”を張り、最新の知識を交換し合い、お互いの書いて表現する能力を強化することを可能にしてくれる。

これからの時代は“書き手パワー”の時代だ。マニュアルの原著者あるいは責任者が、内容、ページレイアウト、グラフィック部分、印刷、配給をすべて管理できる時代である。また、文字や絵が無造作にデータベースから出入れされ、コミュニケーション・ネットワークの間を縦横無尽に飛び回る時代である。こういう時代がくると、技術の変化は、ほとんど自動的にマニュアル作成作業に反映され、そこから得られるノウハウが、さらにチュートリアル・プログラムの作成に役立てられることになるだろう。

書き、描き、出版するというテクノロジーのほとんどすべての進歩によって、原稿を書き換えるのに必要なコストや手間は、どんどん軽減されていく。ほんの数年前には、驚くほど高価で時間のかかった校正作業も、今では安価でしかも早くできる。したがって、数年後には、本書が提唱しているような守りに徹したややかたくるしい方法は、比較的大がかりなマニュアルにのみふさわしいものとなるだろう。ちょっとしたマニュアルについては、それを作り、テストし、作りっぱなしにしたとしても、その費用はたいしたことはなくなり、モデルを作ってそれをテストするまでもなくなるだろう。

新しいテクノロジーは、徐々にマニュアルの質や使いやすさを高めるばかりでなく、その短期的コストや作成期間も削減してくれるだろう。事実、こうした傾向は、マニュアルの品質向上を妨げる2大要因を凌駕するものである。2つの要因とは、直接的な出費を近視眼的な利益で見ようとすることと、プロジェクトの期限をせっかちに先取りしようとすることである。言い換えれば、使いやすいマニュアルなら、必ず出費に見合う利益をもたらしてくれるものだが、第五世代においては、そうした原価償却期間は、劇的に短くなるだろう。

要するに、情報処理テクノロジーの次世代は、われわれに書きたいように書き、描きたいように描く能力を与えてくれるだろう。われわれは、こうした可能性に向けて着実に歩んで行くだろう。われわれはこの並外れたパワーを William Zinsserのいう「うまく書くための3大目標」*すなわち、明確さ、簡潔さ、そして人間らしさを追求するために利用したいと願うものである。

原注＊ William Zinsserの「うまく書くための3大目標」：「Writing With a Word Processor」（New York：Harper Colophon Books,）1983, p.112

付録

A. “構造化”マニュアル目次の実例

以下の例は、本書に書かれているアウトライン作成の趣旨を実証したものである。(もちろんこれらのアウトラインは、そのまま目次として使用される。)

図表 A.1　ライン・エディター、通称ライン・エドのオペレーター・ガイド

1.　本マニュアルの読み方
2.　２つの機能：入力と編集
3.　ライン・エド入力の仕方：LEDIT について
4.　ライン・エドコマンドの入力方法
　4.1　カーソル/ポインターの動かし方
　4.2　新規データを入力する(input)
　4.3　追加データを挿入する
　4.4　２つのデータを結合する
　4.5　既存のデータを呼び込む
　4.6　不要なデータを削除する
　4.7　データを移動させる
　4.8　ファイル内の“全体的”変更をする
5.　ライン・エドの出力管理
　5.1　入出力のフォーマットをする（３つの方法）
　5.2　印刷プロトコルを指定する
6.　削除した項目を再生する
付録：メッセージ一覧

図表 A.2　ワープロ・システムの事務管理者用ガイド

1.　文書処理部門の役割分担
　1.1　認可：誰が仕事を承認するか
　1.2　仕事の優先順位をどう決めるか
　1.3　組織体系一覧
　1.4　義務および職責一覧
　1.5　ワープロ機能使用のための８つの必須条件
2.　ワープロ装置の起動法
　2.1　メイン処理装置を動かす
　2.2　プリンターを動かす
　2.3　通信路を開く
3.　第一段階処理：基本的な文書処理
　3.1　文書ファイルを定義する
　3.2　テキストを入力する
　3.3　校正用原稿を印刷する
　3.4　テキストを編集する
　　3.4.1　第１稿でのもっとも一般的な20の誤り
　　3.4.2　もっとも難しい３つの校正
　3.5　最終原稿を印刷する
　3.6　最終原稿を電子メールで送る
4.　第二段階処理：複雑で専門的な文書
　4.1　旧文書群から文書を集める
　4.2　文書変数を結合する
　4.3　ワープロソフトで計算処理をする
　4.4　郵送/配布ファイルを作る
　4.5　曖昧な入力を解釈する（デフォルト・ルール）
5.　方針：全文書の収集と記憶
6.　方針：クライアントの信頼を保持する
7.　方針：著作者と管理者層の圧力に屈しない

図表 A.3 専用電話ネットワーク作成マニュアル(抜粋)

第3章 テリトリー・マップを構築し実証する
　3.1 各テリトリー・セットは電話データベースである
　3.2 新規テリトリー・セットの開始の仕方
　3.3 1セクターのテリトリー・セットにデータを入力する
　3.4 AZ-60 から AZ-190 へセット・データを転送する
　3.5 補助データベース(INTERIM)を作成する方法
　3.6 INTERIM からテリトリー・セットへデータを読み込む
　3.7 グラフ・マップへのアクセスの仕方
　3.8 マップ全体に関する5つの必要条件
　3.9 システムにマップを“導入”する方法
　3.10 マップ・データを効率よく入力する手順
　3.11 グラフ・マップの ZOOM 機能の使い方
　3.12 マップの操作方法
　　　1 ビルとノードを追加する
　　　2 ビルとノードを変更する
　　　3 端末データを追加する
　　　4 端末プロファイルを変更する
　　　5 端末/設置場所のマトリックスを修正する
　　　6 2つのノード間にセグメントを追加する
　3.13 入力データの確認の仕方
　3.14 点検:テリトリー・セットのチェックリスト

図表 A.4 電子メール・システム(E-POST)のユーザー・ガイド

1. 本マニュアルで使用される規約
2. 通常のメールの代わりに E-POST を使う3つのメリット
3. E-POST メッセージをいつ受け取ったらよいか
4. E-POST にアクセスする
5. E-POST からヘルプを得る
6. E-POST を利用する
7. E-POST のインデックスを利用する
　7.1 インデックスにアクセスする
　7.2 姓に U のつく名前を検索する
　7.3 口座番号や暗証番号によって U のつく名前を検索する
　7.4 他の OGR コンピュータから U のつく名前を検索する
8. 送信リストを利用する:監視
　8.1 送信リストをセットアップする
　8.2 送信リストを変更する
　8.3 送信リストで E-POST メッセージの送り先を決める
9. E-POST メッセージを印刷する:3つの方法
　9.1 画面を印刷する
　9.2 STORE 機能を使って印刷する
　　9.2.1 E-POST メッセージを STORE として保管する
　　9.2.2 STORE ファイルを印刷する
　9.3 ワープロ機能を使って印刷する
　　9.3.1 E-POST メッセージを文書として保管する
　　9.3.2 SCRIBE-15 に文書を送る
　　9.3.3 SCRIBE-15 の文書番号によってメッセージを印刷する
10. 既存のファイルを E-POST メッセージとして送る方法
11. サブ・インデックスに E-POST メッセージをファイルする方法
12. E-POST コマンド一覧

B. 本書において使用される用語抜粋

下記の表は、本書において使用されているいくつかの重要な用語を定義したものである。

用語	定義
アクセス容易性 (accessibility)	マニュアルからの情報の入手、あるいは引出しが容易にできる様子
エンジニアタイプ (engineer stereotype)	草稿執筆にはあまり努力をせず、分析や設計にほとんどの力を傾けるような書き方
可読性 (readability)	処理の推移に沿って読み進むことのできる様子、難易度順に書かれることが多い
可用性 (availability)	マニュアルおよび必要情報の有無
教育用モジュール (instructional module)	初心者を対象として、ある1つの操作方法や概念などを新たに解説するマニュアルの部分
業務別 (task-oriented)	ユーザーの携わる実務に即して考案されている〜
芸術家タイプ (artistic stereotype)	分析や設計にはあまり力を入れず、原稿執筆にそのほとんどの努力を費やすような書き方
構造化 (structured)	そのプロセスがトップダウン（下方展開型）分析やモデルの作成によって考案された〜、その構造がいくつかのモジュールとその組合わせになっている〜
構造化アウトライン (structured outline)	構造化されたマニュアル内の各モジュールに付けられたヘッドラインのリスト
構造上の欠陥 (structural error)	マニュアルの内容をもっとも読みやすい順番に構成する上での誤り
GOTO	無条件分岐；マニュアルにおいては、流れの途中で他のモジュールに出入りすること
項目／対象者のマトリックス (topic/audience matrix)	マニュアルに書かれるべき項目と対象となるユーザー(読者)を縦横に配列した一覧表で、マニュアルの構成を考えるのに使用される
信頼性 (reliability)	正しい運用の妨げとなったり、ユーザーに誤解を与えたりしない様子
ストーリーボード (storyboard)	マニュアル内の各モジュールの仕様書や、取り上げられているマニュアルのモデルを掲示する作業板
対象読者 (audience)	共通の使用目的（類似した業務内容）と共通の経験を持つ読者層
適合性 (suitability)	マニュアルがいかにユーザーの使用目的にマッチし、その業務をどこまで支援できるかという度合い
デモンストレーション・モジュール (demonstration module)	熟練者を対象として、ある処理の全過程を解説することを目的としたマニュアルの部分

動機付けモジュール (motivational module)	読者があまり実行したがらない事柄を実行させるよう仕向けるマニュアルの部分
表現上の失策 (tactical error)	マニュアルの草稿を、より明白に、より読みやすく編集する上での誤り
ヘッドライン (headline)	あるテーマあるいは実際的な表現が盛り込まれた表題で、マニュアルの各モジュールに付けられる
方針上の失敗 (strategic error)	マニュアルと情報製品を正しく分析し、再構築する上での誤り
マニュアルセット (documentation set)	マニュアルおよび他の情報製品の全体を、対象となるシステムに関連付けて定義した企画案
メンテナンスのしやすさ（保守性） (maintainability)	システムあるいはマニュアルをバグ取りしたり、修正・改良したりしやすい度合い
モジュール (module)	システムあるいはマニュアルにおいて、ある1つの機能を果たす独立した最小単位
モデル (model)	システムあるいは製品のテストを容易にするために用いられる設計図
ユーザー用文書 (user documentation)	ユーザーがある特定の技術を応用することを助けるよう考案されたあらゆる文書（情報製品も含む）
有用性 (usability)	システムや製品、マニュアルの使いやすさ
有用性の指標 (usability index)	マニュアルの使いやすさの度合い。読者が読飛ばしや枝分かれ、回り道をする回数が多いほど、そのマニュアルは読みにくいものとなる
リファレンス・モジュール (reference module)	予備の覚書きとして、後でユーザーが“参照する”のに役立つマニュアルの部分
redundancy	読者の負担を軽減し、混乱せずに集中できるよう配慮して、繰り返し解説すること

C. ドキュメンターのための参考文献

マニュアル作成に関する研究は確かに新しい分野だが、今までにもまるっきりなかったわけではない。決して多くはないが、充分役に立つ書籍や定期刊行物のライブラリを次に挙げよう。これらは、駆出しのドキュメンターにとって（いや、ベテランにとっても）非常にためになるものであろう。

まず推薦図書をアルファベット順に紹介するが、その前に特に価値ある1冊を挙げておこう。

Sandra Pakin & Associates. *Documentation Development Methodology.* Prentice-Hall, 1984.

この極めて有益な著作は、駆出しのライターにうってつけの1冊である。また、文書作成に関して基準を持っていない会社は、この本の内容を基準や作成手順のマニュアル（手引書）として取り入れることもできるだろう。

Pakin の著書に加えて、他にも以下のようなマニュアル作成に関する有用な文献がある。

d'Agenais, J., and J. Carruthers. *Creating Effective Manuals.* South-Western Publishing Company, 1984.

Grimm, Susan. *How to Write Computer Manuals for Users.* Lifetime Learning, 1982.

Horn, Robert. *How to Write Information Mapping.* Information Resources Inc, 1976.

Kelly, Derek. *Documenting Computer Application Systems.* Petrocelli Books, 1983.

Matthies, Leslie. *The New Playscript Procedure.* Office Publications, 1977.

Rubin, M.L. (ed.). *Documentation Standards and Procedures for On-Line Systems.* Van Nostrand Reinhold, 1979.

Schoff, G., and P.Robinson. *Writing & Designing Operator Manuals.* Lifetime Learning, 1984.

Zaneski, Richard. *Software Manual Production Simplified* Petrocelli, 1982.

マニュアル作成に起用された人々の多くが、専門職としてのテクニカル・ライティングの分野では新参者である。以下の本は、特に技術的および科学的な主題を、いかに明快に書くかに関するもっとも定評ある著作の一部である。

Brogan, John. *Clear Technical Writing.* McGraw-Hill, 1973.

Michaelson, Herbert B. *How to write and Publish Engineering Papers and Reports.* ISI Press, 1982.

O'Rourke, John. *Writing for the Reader.* Digital Equipment Corp. No. DECOO-XWRDA-A-D, 1976.

Skees, William. *Writing Handbook for Computer Professionals.* Lifetime Learning, 1982.

Strunk, W., and E. B. White. *The Elements of Style,* 3rd ed. Macmillan, 1979.

Van Duyn, Julia. *The DP Professional's Guide to Writing Effective Technical Communications.* Wiley, 1982.

Weiss, Edmond. *The Writing System for Engineers and Scientists.* Prentice-Hall, 1982.

マニュアル作成に携わる人々は、たいていの場合、そのシステムに関する他の文書（主に企画書、仕様書、報告書など）も書かされることが多い。下に挙げた書物は、コンピュータに関するプロジェクトの管理・運営に必要な文書について語っている。

Harper, William. *Data Processing Documentation,* 2nd, ed. Prentice-Hall, 1980.
London, K. *Documentation Standards.* 2nd ed. Mason & Lipscomb, 1974.
Long, Larry. *DP Documentation and Procedures Manual.* Reston, 1976.
National Bureau of Standards/Institute for Computer Sciences & Technology
Computer Model Documentation Guide (500-73)
Proceed's of the NBS Software Documentation Workshop (500-94)
Guidelines for Documentation of Computer Programs and Automated Data Systems (FIPS PUB 38)
Poschmann, A. Standards and Procedures for Systems Documentation. AMACOM, 1983.
Sides, Charles H. *How to Write Papers and Reports About Computer Technology.* ISI Press, 1984.

さらに、データ処理やコンピュータ科学に関する経験のない人々は、「Computerworld」誌に毎週掲載される"In-Depth"のコーナーを読む習慣を持つことをお勧めする。

製造業に携わるもっともクリエイティブな頭脳労働者たちが、彼らの最新のアイデアや書きかけの本の一部を、そのコーナーで紹介している。

マニュアル作成に関する記事は、いたるところから（Sunday newspapersからも）飛び出てくる。しかし、以下の定期刊行物は、この主題について定期的に書いている。

FOLIO, published by Pakin & Associates in Chicago
**(The SIGDOC Newsletter),* published by the Special Interest Group on Documentation of the Association for Computing Machinery (ACM) in New York
Technical Communication, the Journal of the Society for Technical Communication, Washington D.C.
IEEE Transactions on Professional Communication
"Simply Stated", published by Document Design Center, Washington, D.C.

おそらくもっとも読みやすい論文や記事の集成書は、年刊の「Proceedings of the International Technical Communications Conference」だろう。この選集はSociety for Technical Communicationsを通して手に入れることができるもので、テクニカル・コミュニケーション全般に関するもっとも刺激的で有益な本である。またマニュアル作成に限定して読んでもおもしろい。

D. ドキュメンターのためのソフトウェア・ツール

マニュアルのライターは、自分の属する文書処理や情報処理の部門で仕事をするために直接必要となる技術や技能を、直ちに身に着けなければならない。これは、マニュアルの品質を向上させるだけではなく、ユーザーやシステムやマニュアルに関する問題を見抜く洞察力を彼らに与えることにもなる。

●言葉やテキストに関する製品

・本格ワープロ・パッケージ：テキスト全体の移動（カット・アンド・ペースト）や書式設定（ページレイアウト）ができる。ワープロ・プログラムのもっとも進んだ機能は、フッティング、変則的なページ・ナンバリング、プリント・スプーリング（文書印刷中に他の文書を編集できる機能）などである。

・タイプセッティング・インターフェイス：ワープロの文書ファイルを写植用の磁気命令変換する。こうすればテキストを再入力する必要がなくなる。

・レタリング・プログラム：表題や図表をプリントしたりプロットしたりする際、用意されているフォントを大きいサイズのキャラクタに変えることができる。(このプログラムは、プリンターに装着できるよう、マイクロ・チップとして購入することもできる。)

・アウトライン作成ツール：最近急成長を見せているプログラムの1つで、執筆者が自分のアイデアを練ったり、そのアイデアをさまざまな階層構造や順番に構成したりするのに役立つ。

・テキスト管理機能：テキストをコード化し、データベース内に収納する。こうしておけば、ある特定の筋道の手順を再現したり、“モジュール”を引き出したりするだけでマニュアルを“書く”ことができる。

・いわゆる著作ツールあるいは言語：コンピュータベースのトレーニング・プログラムやオンライン・チュートリアルを作成するのに役立つソフトウェア製品。

・編集支援プログラム：スペル・チェックはもとより、場合によっては文体や用法の問題をチェックしたり、読みやすさの点数を計算することもある。

●グラフィックやアートに関する製品

・ビジネス・チャート・パッケージ：統計データからちょっとした折れ線、棒、円グラフなどを作成するのに使用できる。

・サイン・メーキング・パッケージ：おもに文字で構成されるタイトルページその他の提示物（スライドや透かし文字も含む）を作成するよう設計されたもの。

・ダイヤグラム・ツール：フローチャート、データ・フロー・ダイヤグラム、組織構造図、ガント・チャート、その他ビジネスやデータ処理の分野で頻繁に使用されるダイヤグラムを作成する。

・ドローイング・アンド・ペインティング・プログラム：フリーハンドの作図を可能にし、それを磁気媒体に保存したり、プリンターやプロッターに出力したりできる。これらの製品は、たいてい標準印字辞書を持っており、さまざまな形状で使用できる。

・“クリップ・アート”ライブラリ：印字やイラストなどのライブラリで、ビジネス文書などにプリントあるいはプロットすることができる。たいていドローイング・アンド・ペインティング・プログラムの拡張機能として供給される。何百ものクリップ・アート

を搭載したディスクを購入することは、商業デザイナーに頼んで1〜2枚描いてもらうより安上がりなことが多い。

・**グラフィック強化プログラム**：シンプルなビジネス・チャートやダイヤグラムを見ばえのいいものにしたり、改良したりするプログラム。平面図を立体化することが多い。

●ユーティリティおよび生産性向上に関する製品

・**プロジェクト・マネジメント・パッケージ**：マニュアル作成の管理者が、マニュアルや他の情報製品の開発に必要なスケジュールを組んだり、予算をたてたりするのを助ける。モジュール化の方法でマニュアルを設計・作成するなら、プロジェクト・マネジメントシステムのあらゆる機能を大いに活用することができる。つまりクリティカルパスによるスケジューリング、リソース（人員・資材）の均一配分（山崩し）、長期予測など。

・**キー定義プログラム**：ドキュメンターがキーボードのキー機能を変更できるようにする。このプログラムは、キーボードのキーの再定義（標準キーボードをそっくり変えることも）をすることができるが、もっと重要なことは、ごく少数のキーに50〜60の機能を持たせることができるということである。さらに“マクロ定義”（長い操作手順を2、3のキー操作でできるようにする機能）やよく出てくるフレーズ（繰り返し使用される名詞や用語）をコンピュータの辞書に登録したりすることもできる。これらのユーティリティ・プログラムは、延々と続く気の滅入るような労力を節約してくれる。

・**スプレッドシートおよびファイル管理プログラム**：最後に、もっともポピュラーなビジネス・プログラムだが、これらはドキュメンターにとっても非常に有効である。スプレッドシートは、スケジューリングや財務計画に使用できるだけではなく、どんなマニュアルが必要かを分析するための項目／対象者のマトリックスを作成するのにも役立つ。ファイル管理プログラム（あるいはデータベース管理システム）は、基本的に文書ファイル、特にメンテナンスが必要となるものを保管するためのものである。さらに、ほとんどのファイル管理プログラムが、ストーリーボード作業のために、モジュール・スペックを保存し、印刷し、操作するツールとしてプログラミングすることができる。

索　引

【た行】

【な行】

【は行】

【ま行】

【や行】

【ら行】

【わ行】

訳者あとがき：マニュアル作成の機械化をめざす

現代はまさに「マニュアル化社会」である。コンピュータ・テクノロジーの発達により、マニュアルは専門技術文書から、予備知識のないエンド・ユーザーが手にする文献となった。しかし今までは、マニュアル作成を“エンジニアリング”としてとらえ、明確な方法論を提示してくれる「教科書」はないに等しかった。本書はテクニカルライターをはじめ、技術文書の作成に携わるあらゆる人々に確かな指針を与えてくれる“福音書”である。本書はコンピュータ・プログラミングの方法論である「構造化分析」「構造化設計」をドキュメント作成に応用したものである。さらに、テクノロジー開発の次世代において、マニュアル作成がいかにコンピュータ化されていくかを“見越し”た優れた「序説」でもある。

●本書はテクニカルライターの“福音書”である

本書はEdmond H. Weissの“How to Write a Usable User Manual”(ISI Press, 1985)の全訳である。「使いやすいユーザー・マニュアルの書き方」とでも訳すべきだろうが、著者も序文で語っている通り、テクニカルライターのみならず、**マニュアルをはじめとするあらゆる技術文書の作成に関わる人々全員に捧げられた“福音書”**であるという思いを込め、あえて「マニュアル・バイブル」という大胆な邦題を付した。その名に恥じぬ内容であると確信している。事実わが社ではすでに2年前から本書の方法論にのっとって、パソコン用ソフトウェアのマニュアルをはじめとするさまざまなユーザー用文書の作成を手掛け、単なる机上の空論ではない本書の実践面での有効性を立証し、クライアントやエンド・ユーザーから高い評価を得ている。さらにわが社では本書の方法論をもとに、パソコン用データベース言語を用いて“マニュアル・プランニング・ツール”を開発し、実務に応用している（ツールについては後で述べる)。

われわれのようにマニュアルやその他のユーザー用文書の作成を任とする者らにとって、かつてこれほど骨太な、学問的体系にのっとった、しかも確かな指針を与えてくれる極めて実践的な「教科書」があっただろうか。いわゆるマニュアル作成に関する手引書の類はあるにはあったが、しょせん“つづりかた”の域を出ない枝葉末節の部分を扱ったものばかりで、手本となるような優れた先達の業績があるわけでもなかった。少なくとも私にとっては、乏しいセンスとあてにならない勘にたよる試行錯誤の連続だったというのが正直な感想である。したがって本書の存在を知り、それを翻訳出版するという幸運に恵まれたことは、私にとって目からウロコの落ちるような体験であり、文字通り本書は“救済の書”となってくれたわけである。事実この本一冊で、わが社の文書作成業務はすっかり塗り変えられてしまった。

●「マニュアル化社会」がやってきた！

世はまさに「マニュアル化社会」である。恋愛の仕方から葬式の出し方まで、あらゆるものが“マニュアル化”されている。日常生活のみならず、企業などの組織的な営利活動においても、CI（コーポレート・アイデンティティ）ブームともあいまって、**マニュアルが単なる商品の付随物ではなく、市場競争に勝ち残る重要な鍵**になるという認識が高まっている。しかしこうしたマニュアルの重要性に関する認識は、何も今に始まったことではなく、われわれの先祖がまだチョンマゲを結っていた時代から、マニュアルは「虎の巻」とか「指南書」という形で庶民の生活に深く根付いていただけでなく、“天下取り”のための情報源としても重要な役割を演じてきたのである。

しかし「虎の巻」とか「指南書」というもはや死語になりつつある言葉が“マニュアル”という言葉に姿を変えて現代によみがえったのには、やはり何といってもコンピュータ・テクノ

ロジーの発展が大きく関与しているといえるだろう。コンピュータという怪物の出現によって火の点いた技術革新は、今や日の出の勢いで日進月歩し、それにともなう**システムの複雑化**、**プロジェクトの大規模化**によって、時代はごく小数の天才的なプログラマーが密室の中でブロイラーのようにこつこつと金の卵を生み出し続けていればよかった第一期から、**複数の人間の組織的かつ有機的な共同作業**が要求される第二期に突入している。こうした共同作業を可能とするためには、天才的なひらめきや勘にたよるやり方ではなく、**万人に理解し得る体系的で普遍的な基準・ルール・指針**というものが必要であり、複数の人間が同時に動く以上、スケジュールやコストを管理するプロジェクト・マネジメントの手法とも結び付かなければならない。

こうした状勢の変化から、マニュアルの作り方もおのずと変わってきたといえる。専門職であるプログラマーやオペレーターが理解できればよかったマニュアルが、今ではテクノロジーの開発とは無縁な家庭の主婦や子供達までが手にする文献となったのである。しかし実際の作成過程はどうか？“マニュアル（manual)”という言葉はもともと「手造りの、手作業の」という形容詞であり、最近では“オートマチック”つまり「自動の、自動的な」という言葉の反意語として使われるケースが多いが、その語義通り、マニュアル作成の現場は相変わらず“手作業の”域を出ないのが実状である。それは、今までマニュアル作成の機械化を可能にしてくれる明確な方法論がなかったからにほかならず、おそらくは本書の出来を待たなければ実現でき得なかったことだろう。

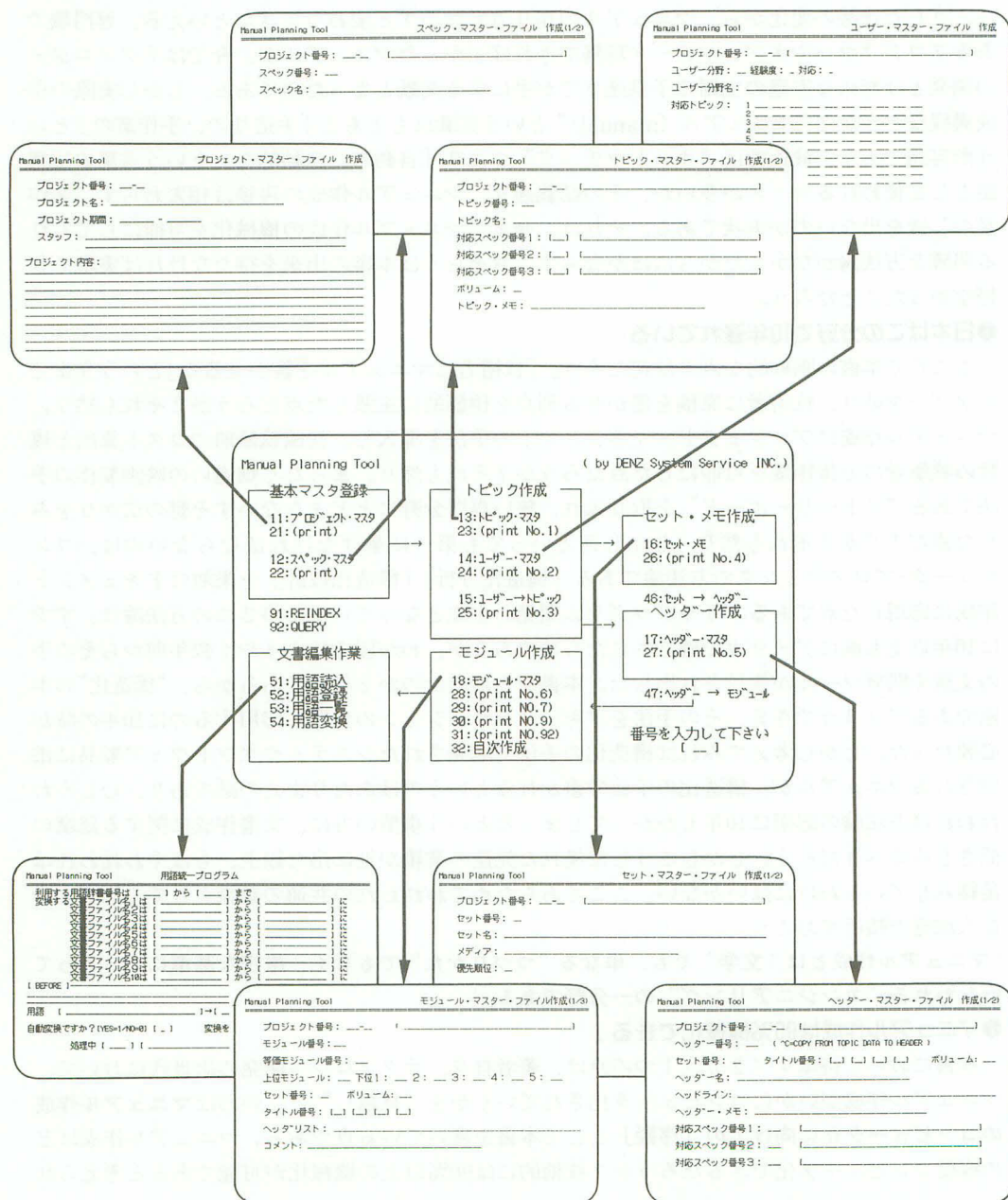
●日本はこの分野で10年遅れている

ところで本書の画期的な点とは何だろう。「技術者にマニュアルを書かせるな」という今までのタブーを破り、技術者に原稿を書かせる利点を積極的に主張した点だろうか？それも然り。マニュアル作成にプロジェクト・マネジメントの手法を導入し、初稿執筆前のコスト算出と複数の執筆者の分担作業を可能にした点だろうか？それも然り。まったく畑違いの映画製作の手法である“ストーリーボード”を取り入れ、狭い専門分野にとどまらないすそ野の広がりをみせた点だろうか？それも然り。しかし何といっても第一に挙げなければならないのは、コンピュータ・プログラミングの方法論である「構造化分析」「構造化設計」を果敢にドキュメント作成に応用した点である。今やプログラム開発の主流となっているこの２つの方法論は、すでに10年以上も前にアメリカで確立されたものであるが、わが国ではようやく数年前からその手の文献や開発ツールが注目され出した。本書が世に出たのが２年前であるから、“構造化”の本家であるアメリカでさえ、その手法をドキュメンテーションの分野に応用するのに10年の時が必要だった。しかし考えてみれば構造化の手法で開発されたシステムやソフトウェア製品に添付されるマニュアルが、構造化の手法で書かれるというのはあたりまえの話であり、むしろわれわれは方法論の応用に10年もかかってしまったという事情の方に、文書作成に関する認識の低さをみるべきだろう。しかしこうした優れた先達の業績が世に出た以上、もはやわれわれは足踏みしているわけにはいかない。ここであらためてわれわれの共通の認識として、次のような大前提を掲げておこう：

「マニュアル作成とは“文学”でも、単なる“つづりかた”でもなく、厳密な基準にのっとって行なわれる“エンジニアリング”の一分野である。」

●マニュアル作成は90％機械化できる！

本書において特筆すべきもう１つの点は、著者自身、テクノロジー開発の次世代において、マニュアル作成がいかにコンピュータ化されていくかを“見越し”て、いわば**マニュアル作成のコンピュータ化に向けての「序説」**として本書を書いている点である。マニュアル作成はどの程度コンピュータ化できるだろうか？技術的には90％以上の機械化が可能であると考えられ

る。事実、最近ではワープロ・電算写植機・デスクトップパブリシングなどの発達により、編集・校正、デザイン、レイアウト、製版、印刷などの工程はだいぶコンピュータ化が進んでいる。しかし「マニュアル作成は“エンジニアリング”である」と謳った以上、特に厳密にシステム化しなければならないのは、分析・設計のプロセスである。今まで一番コンピュータ化が望まれていながら、それを可能とする明確な方法論がなかったために後手に回っていたのがこの２つのプロセスだといえるだろう。そうしたコンピュータ化の困難な２つのプロセスをいかにシステマチックに進めるかを説いたのが本書であるといっても決して過言ではない。

すでにわが社では本書の方法論にもとづき“マニュアル・プランニング・ツール”を開発し、マニュアル作成をパソコンでデータベース管理していると冒頭で述べたが、最後にこのツールについて少し紹介しておこう。

まずこのツールで特徴的なことは、マニュアル作成の対象となる製品の仕様書をワープロまたはデータベースで管理する点である。本書にもあるように、製品の設計がしっかりしていて仕様書が完備されていれば、当然マニュアルも書きやすく、品質も向上し、手間も合理化される。このツールでは仕様書の内容を項目ごとにコード番号を付けてマスター登録し、マニュアルのどのモジュールに該当するかを指定しておけば、モジュール・スペックを書く際、自動的に仕様書の該当項目をモジュール内に読み込むことができる。

このツールは上記の仕様書マスターの他に、プロジェクト管理データを登録しておく「プロジェクトマスター」、マニュアルに含めるべき項目の分析結果を登録しておく「項目マスター」、対象者分析の結果を登録しておく「ユーザーマスター」、モジュール・スペックの内容を収めた「モジュールマスター」などのマスターファイルから構成されている。これらのファイルが有機的な関係で結合され、「項目リスト」、「対象者リスト」、「項目／対象者のマトリックス」、そして分析のプロセス全体の報告書である「マニュアルセット・メモ」を自動的に出力することができる。

これらの分析結果をもとに、本書の示す設計の手順にそって、まず項目リストを従来型のアウトラインに並べ換え、ヘディングの“文体”をヘッドラインの形に書き換えて独立アウトラインを作り、それをさらにモジュール・サイズに分解して構造化アウトラインを作成する。こうして項目リストはモジュールリストにコンバート（変換）されたことになる。

モジュールにはすでにヘッドラインが付けられているので、要約と図表の説明を書いて打ち出せば、モジュール・スペックが出来上がる。前述したように製品の仕様書からそのモジュールの内容に該当する項目を読み込むことができるので、それをそのまま注釈としてスペックに打ち出すこともできる。

ストーリーボード作業を終了したら、その結果をモジュールリストに反映させ、設計を「凍結」させる。また、アウトラインが確定した時点で各モジュールに階層を表わす番号を付ければ、マニュアルの「目次」および版下製作や印刷の際に必要となる「台割」をページ番号付きで自動的に打ち出すこともできる。

以上のように、マニュアル作成システムの基本的な骨組はすでに出来上がっているため、これをもとに、わが社ではさらに総合的なシステム開発に取り組んでいる。技術的には不可能なものではないため、そう遠くない将来、マニュアル作成の90％システム化を実現できるだろう。

本書は、実に多くの人々の協力によって翻訳出版にこぎつけたものである。特に社長みずから陣頭指揮に立って翻訳協力にあたってくださったフランス語情報センターの中平信也氏、まったくの無報酬で骨惜しみせぬ協力をくださり、さらにプロの翻訳家としての目を拙訳に注いでくださった浅岡伴夫氏、スケジュールの遅れに辛抱強く付き合ってくださった啓学出版の鈴木信行氏および高橋弘子女史に心から謝意を表したい。また、縁の下の力持ちとなってくださった立花芳子女史、岡部保之氏、中川和夫氏、山下寛氏、上野一成氏、戸辺雄一郎氏、染野芳輝氏、秦玄一氏にも末筆ながら深く感謝の意を表し、長すぎるあとがきの筆を置きたいと思う。

1987年８月31日

小　林　　敦

著者紹介

Edmond H. Weiss Ph. D.

フリーのコンサルタント。アメリカ、カナダを舞台に、テクニカル・コミュニケーションおよびドキュメンテーションに関するセミナーで教鞭を取っている。

1942年フィラデルフィア生まれ。University of Pennsylvania にて学士号および修士号を取得。Temple University にてスピーチ・コミュニケーションの博士号を取得。

著者のキャリアは多岐に渡っている。テレビのドキュメンタリー台本、調査研究提案書、トレーニング・プログラム、数え切れないほどのマニュアル、その他の報告書や新聞雑誌記事など、多くの文書を手掛けている。工場や政府での執務経験を経て、現在コンサルティング会社「Crown Point Communications」の代表。

著書：「The Writing System for Engineers and Scientists」(Prentice-Hall, 1982)

訳者紹介

小林　敦（こばやし　あつし）

1958年東京生まれ。早稲田大学法学部卒。アテネ・フランセ教授助手を経、英語・フランス語の技術翻訳に従事。その後専門学校にてプログラミングを学び、デン・コーポレーション入社。同社開発室長を経て独立。現在コンピュータ関連の専門書やマニュアル、雑誌原稿などの企画・執筆・翻訳に従事。最近ではマニュアル作成セミナーの講師としても活躍中。

著書：『Do「スウィング」―ビギナーのためのRDB入門―』（日刊工業新聞社）（共著）

『マニュアル作成の構造化手法』（日経マグロウヒル社）

訳書：『ドキュメンテーション作成方法論』（日経マグロウヒル社）（共訳）

（方法論やツールについてのお問合わせは下記まで）

システム　リーブル

〒101　東京都千代田区神田神保町2-20-2　ワカヤギビル406　Tel：03(234)8130

〈翻訳協力〉

中平信也（フランス語情報センター代表）

浅岡伴夫（テクニカル・トランスレーター）

マニュアル・バイブル
――使いやすいユーザー・マニュアルの書き方――

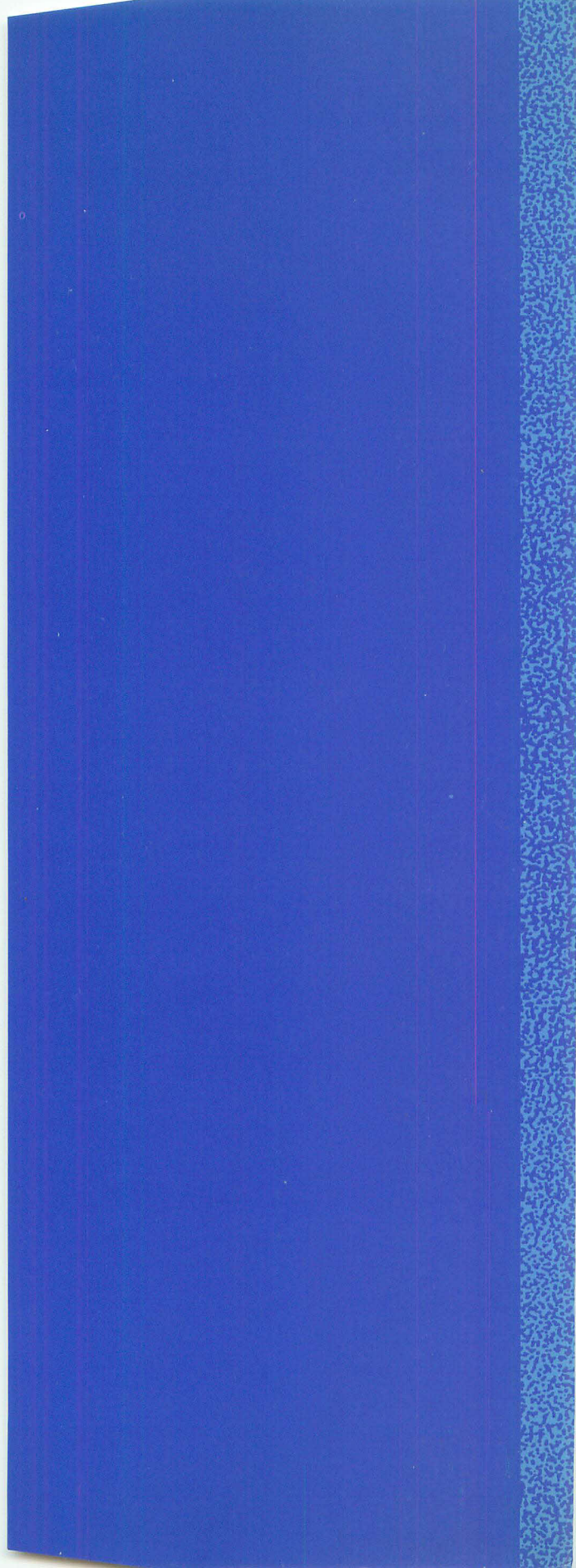
1987年10月31日　第1刷発行
1988年8月15日　第2刷発行

著　者　E・H・ワイス
訳　者　小林　敦
発行所　啓学出版株式会社
代表者　三井数美
郵便番号　101
東京都千代田区神田神保町1-46
電　話　東京03(233)3731(代)
振　替　東京3-109286

印刷／昭和工業写真印刷所
製本／徳住製本所
本書の定価はカバーに表示してあります

ISBN4-7665-0268-X

Printed in Japan

H. T／デン

マニュアル・バイブル
使いやすいユーザー・マニュアルの書き方

エドモンド.H.ワイス・著　小林敦・訳

[出] 1953
啓学出版
定価2890円(本体2806円)

USABLE USER MANUAL

Edmond H. Weiss
Atsushi Kobayashi

ISBN4-7665-0268-X C3055 ¥2800E